ANONYMITY VS. TRACEABILITY: REVOCABLE ANONYMITY IN REMOTE ELECTRONIC VOTING PROTOCOLS

by

MATTHEW JAMES SMART

A thesis submitted to
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
May 2012

Abstract

Remote electronic voting has long been considered a panacea for many of the problems with existing, paper-based election mechanisms: assurance that one's vote has been counted as cast; ability to vote without fear of coercion; fast and reliable tallying; improvement in voter turnout. Despite these promised improvements, take-up of remote electronic voting schemes has been very poor, particularly when considering country-wide general elections.

In this thesis, we explore a new class of remote electronic voting protocols: specifically, those which fit with the United Kingdom's requirement that it should be possible to link a ballot to a voter in the case of personation. We address the issue of *revocable anonymity* in electronic voting. Our contributions are threefold. We begin with the introduction of a new remote electronic voting protocol, providing revocable anonymity for any voter with access to an Internet-connected computer of their choice. We provide a formal analysis for the security properties of this protocol. Next, we are among the first to consider *client-side security* in remote electronic voting, providing a protocol which uses trusted computing to assure the voter and authorities of the state of the voter's machine. Finally, we address revocable anonymity more generally: should a user have the right to know when their anonymity has been revoked? We provide a protocol which uses trusted computing to achieve this.

Ultimately, the work in this thesis can be seen as a sound starting point for the deployment of remote electronic voting in the United Kingdom.

For Gem, Alexis and Rosie, the girls in my life

Acknowledgements

To acknowledge the support, help and guidance of all of the people to whom I owe my gratitude is a rather daunting task. I apologise in advance if I have forgotten you!

My first thanks go to Eike Ritter, my supervisor. His willingness to take me on as a Masters, and then Ph.D. student, brought me to where I am today. His ever-present cheeriness and dedication, his reassurance and guidance, leave me very much in his debt. I am indebted too to several other colleagues at Birmingham—not least Tom Chothia and Guilin Wang, both sources of guidance for my earlier work, and many of the other members of the School's Formal Verification and Security Group. From a non-research perspective, I will treasure the friendships I have shared with not only staff, but also doctoral research colleagues in the School. Achim Jung, Mark Lee, Jon Rowe; our fantastic technical support and back-office staff; Chris Staite and my other fellow regulars at the School's 'cookie breaks': you have all made the occasionally difficult environment of academic research much more enjoyable, and have made Birmingham such a great place to be. Thank you.

Foremost, however, my eternal gratitude goes to my partner, Gem. Her unending love, patience and support throughout—and her ability to make the tallest mountain seem scalable—leave me lost for words. It is without question that I would not have reached this point without her. I don't say it enough, but thank you—so very much.

Thanks too to my family: my parents, Sue and Roy, for giving me the drive to begin my Ph.D. in the first place, and for helping me to believe that I could achieve my goals, however ambitious. Though they may not have been directly involved in my research life, their part in the last five years has been no less important. To James and my other close friends, and to my extended family, of whom there are too many to name: though you may not have realised it, just

by *being there* you have been of inestimable importance.

Finally, I would thank all of those other people who gave me the sheer *love* for what I do in the first place. John Lees, Kevin Dunn, Jane Norton: your passion for teaching in computing gave me the drive to learn more, and to someday achieve the same fascination in my students' faces as you imbued in yours.

"You ought to have some papers to show who you are," the police officer advised me.
"I do not need any paper. I know who I am," I said.
"Maybe so. Other people are also interested in knowing who you are."

—Traven, *The Death Ship*

Contents

List of Figures

List of Tables

1 Introduction and Motivation

> I consider it completely unimportant who in the party will vote, or how; but what is extraordinarily important is this—who will count the votes, and how.
>
> — Josef Stalin

Designing electronic voting protocols is notoriously difficult: so much so, that despite a wide breadth of work in the field, very few electronic voting systems have been deployed on a country-wide basis. Indeed, early forays into electronic voting have resulted in widespread criticism of the election process, and questions as to the reliability and trustworthiness of machines used to process votes (Mercuri, 2002; Cranor, 2001; Jorba et al., 2003; Chaum et al., 2005; Dill et al., 2003). In the United Kingdom, despite considerable research into the deployment of electronic voting, and a number of local trials, it seems unlikely that a practical solution will be deployed in the near future.

The main difficulty in designing suitable e-voting protocols is in the satisfaction of an ever-growing, apparently contradictory set of requirements: foremost, one must preserve the *secret ballot*: i.e., a voter's vote must remain unlinkable to them. Compounding the set of requirements

ascribed to traditional, paper-ballot elections (such as those currently used in the UK), however, is an extra set of requirements, augmented by security researchers over the past twenty years. Requirements that are not enforced in current election practices (such as individual verifiability: the ability to verify that one's vote is counted as cast—a notion which does not currently exist in the UK) form an important part of the criteria by which any new electronic voting protocol is deemed to be acceptable.

Much recent researched has focused on the design of *remote* voting protocols: namely, those that do not restrict the physical location of the voter, instead allowing them to use any Internet-connected machine to vote. The appeal of remote voting lies mainly in the benefit it offers: increased turnout of marginal voters—those that are politically engaged, but unwilling or unable to visit polling stations. However, the cost is clear: a decrease in the trustworthiness of the environment in which one votes means more work is required to satisfy the aforementioned requirements.

Working to increase the 'security' of an electronic voting protocol often increases its complexity. Thus we are left with the juxtaposition of novice end-users, who are unwilling to trust protocols which use complex cryptography, and security researchers, who include said cryptography so that e-voting protocols are trustworthy. As we will discuss, cryptography is a necessary factor in any remote e-voting scheme.

The electoral systems of the United Kingdom (and New Zealand) are somewhat unique, in that they have an unusual legal requirement: it must be possible for an authority to *trace* a voter from their ballot, given the appropriate legal permission (Blackburn, 1995; Jonker and Pieters, 2010). Typically, such permission is given in the case of personation (say, for example, a voter attempts to fraudulently vote on behalf another voter who has died, or is otherwise unable to vote). It is for this reason that in the UK, ballots are numbered:

> Present practice in the UK involves the ballot papers carrying an inconspicuous identification number…The voter number, as given in the electoral register, is recorded on the counterfoil when the voter is given her ballot paper in the voting station (Randell and Ryan, 2005, p. 3)

In keeping with research in surrounding computer security fields, we term the ability to trace vot-

ers *revocable anonymity*. This thesis is concerned with the design of trustworthy remote electronic voting protocols, which provide revocable anonymity.

1.1 Approach

The aim of this thesis is to explore the design and feasibility of *remote electronic voting protocols* which permit revocable anonymity, as required in UK general elections. Here, we provide an outline of the approach we take to solving this problem.

We begin with an extensive analysis of the problem domain: first, we explore the requirements which we strive to satisfy, then discussing much of the most important work in electronic voting in recent history. We address protocols designed under a number of methodologies, including paper-based, "end-to-end verifiable" protocols. We then proceed to explore several research questions, leading to a number of contributions:

Remote Electronic Voting with Revocable Anonymity Above, we noted that one driving factor of electronic voting is the ability for voters to participate remotely, over the Internet. One expected benefit of this is increased overall turnout. However, the ability to vote remotely is seemingly in direct contrast with the requirement for one's vote to remain private, and for voters to remain uncoercible. In the UK, we have a further requirement: the ballot must be in some way linkable to the voter, given the necessary authority. Our primary contribution is the design of two remote election protocols, which permit revocable anonymity: the first protocols to provide a practical manner in which to do this, without any extra hardware requirements.

Integrating Trusted Computing with Electronic Voting A particular problem with many existing election protocols is that they trust the state of the machine the voter uses. What if the machine is compromised? It then becomes the 'weak link in the chain', allowing man-in-the-middle style attacks on the voter. Our work considers the introduction of *trusted computing* (specifically, use of the TPM) in order to assure the trustworthiness of an unknown, remote voting client. Again, we are the first to consider this and to provide a detailed protocol specification.

Auditable Anonymity Revocation An interesting sociological issue arises from revocable anonymity: should a user who is traced be able to determine that this is the case? Continuing from our work on the use of trusted computing in electronic voting, we explore the issue of how a voter might be notified if her anonymity is revoked. We detail a protocol which uses the TPM to provide such assurances, and which in fact could be used for any area of computer security in which revocable anonymity is an issue.

Formal Protocol Verification We extensively verify our work using ProVerif, an automated reasoning tool based upon the applied pi calculus, drawing on the work of several authors in the field of formal verification. We formalise a number of our requirements in the language, testing that each is satisfied.

1.2 Thesis Organisation

This thesis is organised in a further seven chapters, as follows.

Chapter 2: Background Information We begin with a summary of the requirements and properties which we wish to satisfy in our work. This is followed by a detailed discussion of electronic voting protocols and systems from the past 25 years. We adopt the approach of dividing the protocols according to the cryptographic primitives on which they are based: blind signatures, mix networks, and homomorphic encryption and tabulation of votes. We separately address paper-based protocols such as Prêt-à-Voter, which often use a combination of these preliminaries, but adopt them in a markedly different manner.

Chapter 3: Preliminaries In Chapter 3, we discuss the cryptographic primitives and notation which we use, as well as elaborating on some other preliminaries: namely, trusted computing and the direct anonymous attestation protocol.

Chapter 4: Revocable Anonymity in Electronic Voting In this chapter, we discuss the first of our protocols, which integrates remote, coercion-resistant and verifiable electronic voting with

revocable anonymity. We believe this protocol to be the first work which achieves these properties, and thus the first to be suitable to a UK election scenario.

Chapter 5: Using Trusted Computing We next discuss the design of a related protocol, which uses the TPM (in particular, the DAA protocol) to ensure the trustworthiness of a remote client. Again, the protocol provides (optional) revocable anonymity.

Chapter 6: Making Anonymity Auditable In Chapter 6, we explore whether it is possible for a voter whose anonymity is revoked to be informed of this fact. We discuss the related work of a number of authors, and then again use the TPM to produce a solution.

Chapter 7: Formalisation of Security Properties Our penultimate chapter provides an extensive formal security analysis of the protocol discuss in Chapter 4. We adopt the applied pi calculus and the automated reasoning tool ProVerif, in combination with the work of a number of authors on the formalisation of security properties in electronic voting, to prove the security of our first protocol.

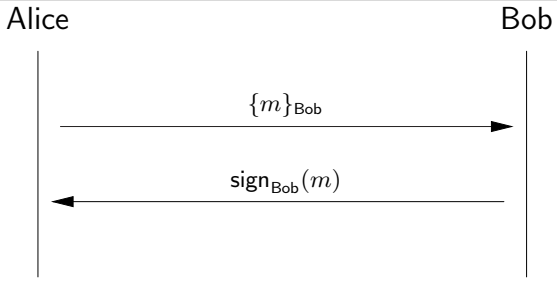
Chapter 8: Conclusions and Future Work Finally, we conclude the thesis, and make comment on future avenues for further work.

Remarks on Notation

In this work, as with much other work in computer security, we adopt the convention that the legitimate, honest voter is female. Specifically, she is denoted Alice. Where necessary, a non-specific party that Alice communicates with is named Bob. Other named entities are generally referred to in a sans-serif typeface, or with a single letter, *viz.* $\mathbb{A}$. Cryptography and mathematical calculations are denoted by *italicised serif* type, except where we refer to functions that we have already defined, such as the generation of a designated-verifier signature, denoted in sans-serif type, *viz.* $\mathsf{DVSign}_{\mathsf{Alice}\rightarrow\mathsf{Bob}}(m)$. The only exception to this is where we discuss commands in the TPM's application programming interface, which are denoted in slab-serif text, *viz.* `TPM_Quote`.

Finally, we adopt a "message sequence chart" approach when protocols by way of a diagram. In the figure below, the protocol being represented involves the encryption of a message m with Bob's public key being sent to Bob. Bob replies with his signature on the plaintext m.

Figure 1.1 An example protocol diagram



2 Background Information

Chapter Overview

In Chapter 1, we introduced the topic of this thesis, and the motivation for its completion. In this chapter, we continue to introduce the thesis by way of a detailed discussion of relevant background material. For a thesis combining remote electronic voting with revocable anonymity, there is a considerable amount of relevant material.

We begin with a discussion of electronic voting, starting with a discussion of the most important requirements of a remote electronic voting protocol, covering many of the important aspects of recent electronic voting protocols, and summarising the reasons why these protocols are not suitable for our aims. We then discuss the many ways in which anonymity and revocable anonymity are enforced in security protocols.

Revocable anonymity is not a notion that has previously been considered with regard to electronic voting. However, it has been addressed in depth in other areas, such as electronic commerce, where there is a wide breadth of work. We discuss this work in depth, focusing on a

number of important works, and then finish the chapter with a discussion of Trusted Computing and the Trusted Platform Module, which are of critical importance to us for one of the protocols presented later.

2.1 Remote Electronic Voting with Revocable Anonymity: Requirements

Before we begin to discuss previous work in electronic voting, we discuss, in Table 2.1, what we consider to be the most important requirements for any new remote electronic voting protocol, having considered a wide breadth of work in the field. For each requirement, we give a brief description.

The three properties *voter privacy*, *receipt-freeness* and *coercion-resistance*, which we can loosely group as all being privacy-related, are increasingly strong forms of the requirement that a voter should not be linkable to her vote. Intuitively, however, some of these properties present an obvious conflict. A voter needs to be able to identify that her vote was counted as cast (Individual Verifiability). Yet, how can she do so without gaining enough information to also prove to a coercer how she voted, thereby breaking Voter Privacy (and its related properties, Receipt Freeness and Coercion Resistance)? One approach, as we discuss later, is for the authorities to produce a proof which convinces the voter, but no-one else—by virtue of the fact that the voter could produce the proof herself. Remote Voting presents a clear clash with Coercion Resistance: it is far more difficult to prevent a voter proving to a coercer how she is voting when she is not physically isolated in a known, trusted location. Further, as Chevallier-Mames et al. (2006) note, unless *all* voters on the electoral roll participate in the election, Universal Verifiability is incompatible with Voter Privacy.

The extra properties that we add (revocable anonymity, and coercion resistance in the physical presence of a coercer) further complicate the requirements: if a voter's anonymity is revoked, is only that voter affected? Can she determine that she has been traced? To what extent can we require that an election protocol must be resistant to a physical coercer, standing over the shoulder

Table 2.1 Summary of Properties and Requirements

Property	Description
Correctness and Eligibility	Only eligible voters should be able to vote, and there should be no trace of the protocol resulting in a successfully counted vote, by Alice, for candidate *i*, that did not begin with Alice voting for *i*
Uniqueness	Only one vote per voter should be counted
Receipt-Freeness	The voter should be given *no information* which can be used to demonstrate how they have voted, once voting is complete
Coercion-Resistance	It should not be possible for a voter to prove how they voted or are voting, *even if interacting with a coercer during voting*
"Invisible Absentee" Coercion-Resistance	Strictly a subproperty of coercion-resistance: the voter should remain resistant to coercion even in the *physical presence* of a coercer
Individual Verifiability	A voter should be able to verify that their vote has been counted correctly. Also known as *Voter* Verifiability.
Universal Verifiability	Any observer should be able to verify that *all votes* have been counted correctly. Sometimes worded as "the published outcome is the sum of all votes"
Fairness	No-one can gain any information about the tally (or partial tally) of the election until the end of the voting process
Voter Privacy (Anonymity)	Neither the authorities nor any other participant should be able to link any ballot to the voter having cast it, *unless* the protocol to revoke anonymity has been invoked[1]
Revocable Anonymity	It should be possible for an *authorised entity* to reveal the identity of any *single* voter, by linking her ballot to her
Remote Voting	Voters should not be restricted by physical location (i.e., it should be possible to vote over the Internet)

of the voter? This latter property, invisible absentee coercion-resistance, could be seen to have its roots in the notion of *duress passwords* (Clark and Hengartner, 2008; Stefanov and Atallah, 2010), an idea which has received little direct research attention.

Many electronic voting protocols have claimed to satisfy the above properties. In the next section, we discuss what we consider to be the most important protocols in the field.

[1] Note that voter privacy is never achieved in the strongest possible sense: if all voters were to vote the same way, then all votes are identified. Hence we say that no-one should learn more than that which is obtained from the tally.

2.1.1 The Capabilities of the Coercer

An interesting question to consider is what the coercer, in the context of the properties above, is able and unable to do in our work. We begin by noting that, importantly, the coercer is *not* able to simulate the voter for the *entire* duration of the voting protocol. For us, we guarantee this by requiring that the voter's registration must be done *in-person*—a requirement that is placed upon many related electronic voting protocols. This means that the coercer is not able to witness for certain any of the private information which is given to the voter at the start of the protocol (secret keys, or random values which the voter chooses or is allocated in order to prove the authenticity of her vote). It follows that the attacker can also*not simulate* any party who is responsible for issuing keypairs to Alice—as we will discuss in Section 4.5, such an ability, in many standard electronic voting protocols, would again allow the attacker to simulate Alice entirely.

As a direct consequence, the coercer is never certain of whether any information provided to him by the voter or not is valid (i.e., the voter is 'coerced'), or fake (i.e., the voter is 'cheating'—we define these terms further in Section 7.5.3—by simply claiming that a given value is valid, when it is in fact not). At any point *after* registration, the coercer is able to simulate the voter—but cannot determine whether his vote is counted or not without being certain that he holds the correct private key for a voter. However, we note that the legitimate voter is always able to vote *once* unobserved. This is an assumption widely believed to be the minimal requirement for coercion-resistant voting: if we did not have it, the coercer could trivially simulate all of the voter's attempts to vote, or simply suppress her voting entirely.

The coercer is able to simulate any authority in our protocols, except for the judge (whom we trust out of necessity), and subject to the trust assumptions placed on each of our protocols (see Sections 4.3.2, 5.2.2 and 6.2). We use threshold decryption to ensure that a quorum of collaborating members of an authority group is required to decrypt votes, or generate threshold signatures. If any quorum is of size t for a group size n, we assume that the coercer can corrupt up to $t-1$ members of that group, including himself if appropriate. Where our communication channels are public, the coercer can read a message on any channel, and decrypt it subject to having the correct decryption key. He can intercept any message and later replay it, and can

temporarily block any message (though we assume resilient channels, to achieve liveness). Finally, the coercer is able to inject data into the channels arbitrarily.

2.2 A Brief History of Electronic Voting

In this section, we discuss a number of protocols important to the history of electronic voting. As we will discuss later, these protocols can generally be divided into a number of categories, relating to the cryptography that they use, or the type of election protocol (paper-based, verifiable paper trail, and so on).

Despite a considerable amount of research in the field of e-voting, it has enjoyed little real-world, large-scale success: electronic voting terminals (or DRE—Direct Recording Electronic voting—machines) introduced in the United States have been criticised on numerous occasions, and proven to be insecure (Dill and Castro, 2008; Dill et al., 2003; Bannet et al., 2004; Kohno et al., 2004; Mercuri, 2002; Bannet et al., 2004), as has its attempt at a remote electronic voting protocol for use by the military, SERVE (Jefferson et al., 2004). The British government, however enthusiastic (even stating in 2002 that "by 2011, much of the ground should have been prepared for an e-enabled election" (Local Government Association, 2002, p. 1)), has thus far failed to implement a credible electronic voting solution, but has made some effort in this direction (Storer and Duncan, 2005, 2004). Indeed, one of the only countries considered to have successfully addressed electronic voting in national elections is Estonia, having held national electronic elections in 2005 (Madise and Martens, 2006).

We will consider, in this section, why electronic voting has not been as successful as it arguably should have been, and what can be done to improve uptake. We first begin with a discussion of important developments in the field. Electronic voting protocols can be placed into one of four categories, based on the methods used to record and transmit votes, and to elicit anonymity and verifiability. Earlier protocols are frequently based on the *blind signature* primitive, originally invented by David Chaum for use in digital cash protocols (Chaum, 1982, 1985, 1988). Many more protocols use *mix networks*, again a primitive introduced by Chaum (Chaum, 1981), to

elicit voter anonymity. A third popular technique, eliciting both simple tallying techniques and strong voter anonymity, is in using *homomorphic encryption* to encrypt and sum ballots. Finally, we have a number of *paper-based* and E2E ("end-to-end", universally verifiable) protocols, which either involve the user voting on paper, or having access to some sort of voter verifiable paper trail (VVPAT) as evidence of their vote being cast. Strictly speaking, these protocols often use techniques from the previous three categories. However, because of the substantial differences in how voters vote here, we discuss them separately.

2.2.1 Blind Signature-Based Protocols

Many early electronic voting protocols (and some more recent) are based on the *blind signature* primitive invented by Chaum (1982) (Dini, 2003; Chang and Lee, 2006; Chen et al., 2004). A blind signature is one in which the content of the message being signed is kept hidden from the signer (envision a message being placed in a carbon-lined envelope, and then the *envelope* being signed by someone with no knowledge of the contents). It follows that the "blinding" can then be removed, giving a signed unblinded plaintext message. A typical message m would be signed using a regular RSA signature scheme by calculating $m^d \mod N$, where d is the secret signing key, and N is the public modulus. The typical RSA blind signature would proceed by selecting a random *blinding factor* v, coprime with N as follows for a message m. Then:

$$\begin{aligned} m' &:= mv^e \mod N \\ s' &:= (m')^d \mod N \end{aligned}$$

where e is the public exponent. Note that m' was signed with no knowledge of m, to give s'. The signed, unblinded message can easily be recovered:

$$\begin{aligned} s &:= s' \cdot v^{-1} \mod N \\ &:= m^d \mod N \end{aligned}$$

because $v^{ed} \equiv v$. Note that it is not possible for the signer to link the blinded signature to the unblinded one, without knowledge of v. Some consideration has been given to *fair blind signatures*, which alleviate this problem somewhat (Claessens et al., 2003). Blind signature-based protocols are also inherently incompatible with universal verifiability, since the authorities are generally able to add spurious ballots to the tally (a notion known as *ballot stuffing*).

2.2.1.1 Chaum: Unconditionally Secret Ballots

Chaum's early work on electronic voting suggests a protocol in which the following properties are satisfied:

1. A voter's privacy/anonymity is only violated by cooperation of all other voters
2. Voters can ensure that their ballots are counted
3. Voters wishing to disrupt an election can only cause a small delay before being ejected

In the protocol he details, Chaum provides unconditional security against tracing the senders of messages, and uses blind signatures to do this. The protocol involves a voter, Alice and organisation Admin, and follows the order below for issuing a ballot:

1. Admin broadcasts to all participants:
   - A security parameter s
   - Another integer parameter n
   - An RSA modulus N
   - A prime number $d > N$
   - n random units of the *ring of residue classes* mod N ('units mod N'), v_j where $j \in \{1, \ldots, n\}$
2. Alice sends to Admin $M = (m_{i,j}) : m_{i,j} \equiv v_{\pi_i(j)} r_{i,j}^d$ where $i \in \{1, \ldots, s\}$, with π_i random permutations of $\{1, \ldots, n\}$, and $r_{i,j}$ random units mod N

3. Admin sends back to Alice C, a 'random nonempty proper subset' of $\{1, \ldots, s\}$

4. Alice sends to Admin:

   - An element $k \in \{1, \ldots, s\} \setminus C$
   - $P = (p_{i,j})$ where $p_{i,j} = \pi_i(j)$ for $i \in C$
   - $p_{i,j} = \pi_k^{-1}(\pi_i(j))$ for $i \notin C$
   - $Q = (q_{i,j})$ where $q_{i,j} \equiv r_{i,j}$ for $i \in C$
   - $q_{i,j} \equiv r_{k.\pi_k^{-1}(\pi_i(j))} r_{i,j}^{-1}$ for $i \notin C$

5. Admin verifies that each row of P is a permutation of $\{1, \ldots, n\}$; that $m_{i,j} \equiv v_{p_{i,j}} q_{i,j}^d$ for $i \in C$, and that $q_{i,j}^d \equiv m_{k.p_{i,j}} m_{i,j}^{-1}$ for $i \notin C$

Step 1 above forms only the preliminary phase of the election, and is done only once. Admin also broadcasts an assignment of an outcome to each v_i.

During the *registration* phase, each voter communicates with Admin. If Admin agrees that the voter can register, then voter Alice and Admin conduct the ballot issuing protocol given above. This results in a tuple of n elements $m_{k,i}$, of which the voter selects one, denoted b_l for the l^{th} voter. The final result of the registration phase is the set of b_l. Disputes can still be made at this stage without revealing votes.

Finally, in the *voting* phase, Admin broadcasts the dth roots of all of the b_l values. The lth voter can then recover the dth root on a v_i value by dividing the dth root of b_l by the corresponding $r_{h,j}$. The voter then (anonymously) broadcasts the root of the v_i recovered. The final number of votes for each candidate is the number of dth roots of v_i values corresponding to that candidate (Chaum, 1988, pp. 178–180)

This protocol is, of course, rather complex. More importantly, there is a security risk in only using one Admin—a large degree of trust is placed on this entity, which could misbehave and thereby disrupt the election. This would not result in incorrect results, as voters can verify that their votes are counted, but would lead to the election being voided. As Chaum suggests,

the protocol is robust (but as noted by Fujioka et al. (1993), it is not *fair*, as intermediate election results can be determined by the authorities, nor does it guarantee privacy if the voter complains).

2.2.1.2 FOO, and Related Protocols

Any treatment of electronic voting protocols would be incomplete without a discussion of Fujioka et al.'s protocol (1993), commonly termed *FOO*. Based on the earlier work of Chaum (1988) on blind signatures, it is generally accepted to be one of the first credible (fair, anonymous) electronic voting protocols, and has spawned many descendants.

The FOO'92 scheme includes *voters*, an *administrator* and a *counter* (which could be a public bulletin board). Voters and the counter communicate via anonymous channels (implemented by a mix network, for example). The protocol uses a bit-commitment scheme, a standard signature scheme, and a blind signature scheme. As in the paper, the following notations are used. Note that the authors do not detail how each primitive (e.g., bit-commitment scheme, signature scheme, blinding technique) is implemented:

V_i:	Voter i
A:	Administrator
C:	Counter
$\xi(v,k)$:	Bit-commitment scheme for message v using key k
$\sigma_i(m)$:	Voter V_i's signature scheme
$\sigma_A(m)$:	Administrator's signature scheme
$\chi_A(m,r)$:	Blinding technique for message m using salt r
$\delta_A(s,r)$:	Technique to retrieve a message from a blind signature
ID_i:	Voter V_i's identity
v_i:	Voter V_i's vote

The protocol proceeds in six stages, which the authors briefly define:

1. **Preparation** The voter fills in a ballot, encrypts and blinds it and sends it to the administrator

2. **Administration** The administrator signs the blinded message and returns it

3. **Voting** The voter unblinds the signed encrypted vote obtaining a signed vote, and anonymously sends it to the counter

4. **Collecting** The counter publishes the ballots received

5. **Opening** The voter sends a decryption key for the vote anonymously to the counter

6. **Counting** The counter counts the votes and announces the results

The scheme seems thus far to be quite elegant and simple. The authors now go into more detail about how it works—this detail is summarised below.

Preparation

1. Voter V_i selects a vote v_i and completes a ballot $x_i = \xi(v_i, k_i)$ using a random k_i

2. V_i calculates message e_i using blinding algorithm $e_i = \chi(x_i, r_i)$

3. V_i signs $s_i = \sigma_i(e_i)$ and sends the tuple $\langle ID_i, e_i, s_i \rangle$ to Admin

Administration

4. Admin checks that V_i is authorised to vote. If not, the tuple is rejected.

5. If V_i is authorised, Admin checks that V_i hasn't already voted (applied for a signature)[1]. If V_i has voted, the tuple is rejected

6. Admin checks the signature s_i on e_i using V_i's public key. If it is valid, Admin signs $d_i = \sigma_A(e_i)$ and sends d_i as a certificate of authorisation to V_i

7. At the end of this stage, Admin announces the number of voters who were given authorisation to vote, and publishes the list of allowed $\langle ID_i, e_i, s_i \rangle$ tuples[2]

[1] It should be noted that in this manner, Admin is able to form a list of entities who have voted. This should be avoided if possible, as it is a small breach of anonymity.

[2] Again, this makes the fact that a voter applied to vote public, and is therefore undesirable—the voter's ID should be hidden in some way.

Voting

8. V_i unblinds the received signature of ballot x_i by $y_i = \delta(d_i, r_i)$

9. V_i then checks that y_i is a valid signature on x_i. Else, v_i claims that the signature is invalid by showing $\langle x_i, y_i \rangle$ to Admin[1]

10. V_i sends $\langle x_i, y_i \rangle$ to the counter through some anonymous channel

Collecting

11. Counter C checks the signature y_i on x_i using Admin's public key. If the check is successful, C forms a vote number l and enters $\langle l, x_i, y_i \rangle$ onto a list.

12. Once all voters have voted, C publishes the list, such that all voters will be able to access it, and all voters will be able to verify that each x_i is legitimately authorised by Admin.

Opening

13. V_i checks that the number of ballots on the list is equal to the number of voters (i.e., all voters requesting signatures have to vote). If not, a voter V_i can use his blinding factor r_i to prove this to Admin.

14. V_i checks that their ballot is on the list, otherwise using the $\langle x_i, y_i \rangle$ values they have as proof

15. V_i sends the key k_i with the vote number l to C through the anonymous channel

Counting

16. C opens $x_i = \xi(x_i, k_i)$ using k_i for the vote numbered l, and retrieves v_i. It checks that v_i is a valid vote, then adds the vote to the list

[1] Note that showing this pair does not give V_i's identity if done over an anonymous channel. However, arguably the voter would have to authenticate herself to Admin again to prove that the value of y_i was *meant* to be a signature on x_i and wasn't just, for example, made up. This means that again, V_i's identity is potentially released.

17. The final tally is simply the number of votes for each candidate, which is announced at the end of the counting stage.

This protocol is noted for being particularly elegant, efficient and secure. There is no way for the public to disrupt the election, nor for the voter to change their mind[1]; it is not possible to double-vote, and, because the voter's *ID* value is only given to Admin, there is no way to link V_i and v_i, provided anonymous channels are used to communicate with at least one party. Further, counting is only done at the end of the voting stage (Fujioka et al., 1993, p. 249), meaning that the counter cannot influence the vote by releasing tally information. This means that fairness is maintained. What is more, the system provides no receipt of voting to the voter (merely a proof that their vote is permitted, without detailing what the vote is), meaning that receipt-freeness is maintained. The protocol is not, however, coercion-resistant: neither are many of its descendents.

However, the protocol does suffer from a number of problems. As the authors suggest, if Admin is found to have committed any fraud, then the entire voting process is voided. This is not overly surprising, but is highly inconvenient. Perhaps, if several administrators (with a single list of voters) worked as a group, then a smaller proportion of voters could be asked to recast their votes instead.

The assumption of anonymous channels would presumably involve mix networks, and so cannot be considered an issue. However, the stages in which the election is conducted are problematic. FOO is a three-phase protocol, in which all voters must synchronise at the end of each phase: the *administration* phase must be complete before voters can check the list and vote; the *collecting* phase must be complete before voters can check for their votes on the list and submit their $\langle l, k_i \rangle$ tuples. Note further that voters are actively involved in interaction with the authorities during *tallying* as well as voting, which is particularly inconvenient—voters are, in the current system, apathetic at best; a system requiring them to return twice or more to complete their vote is unlikely to be accepted.

The other issue is that the protocol is not coercion-resistant —an adversary could easily tell a voter to vote a certain way if they were voting remotely, and the bit-commitment scheme means

[1] This presents a problem: one way to avoid voter coercion is to allow a change of mind!

that said voter is then unable to void her previous vote and vote again. Further, as noted by Ray et al. (2001), the encrypted ballot, cast by the voter, contains her signature (meaning that the ballot can be identified). This removes anonymity from the voter. A final point is that no voters are able to abstain—if someone does, then the authorities can conspire to vote on the abstainer's behalf.

Furthermore, as the authors state, the Tallier issues a signed receipt to the voter. This allows the voter to prove *that* they voted, if not how (the paper does not go into detail).

As mentioned earlier, a number of protocols have been spawned from FOO (Herschberg, 1997; Cranor and Cytron, 1997; Foster et al., 2006). Sensus (Cranor and Cytron, 1997) has received particular attention. The protocol claims to solve some of the problems with FOO: the voter does not have to participate in the final tallying stage (and does not need to synchronise with other voters after voting), and the voter can explicitly state that they choose to abstain. As in FOO, it is still possible for invalid votes to be added to the tally by the tallier, who may be dishonest (note that the authors claim that the voter only needs to trust the pollster); further, this is only detected by "any party who checks the authenticity of the validation certificates *for all ballots*". For a large voting population, this is completely unrealistic and would take too much time.

Cranor and Cytron make a number of strong assumptions about their protocol, which draw question to its applicability:

- The authors assume that a vote cannot be traced to its voter by tracing packets sent over the network—hence, an anonymous channel which does not show even the presence of communication is assumed

- The voter is assumed to use a computer in which it is "not possible for clear text messages to be intercepted", hence no part of the system can be 'hacked'

- The authors assume that messages from voters "will not arrive at the validator and tallier in the same order", else unlinkability between votes and voters is clearly violated if the tallier and validator collude

The protocol is a close implementation of FOO, adapted for a real-world scenario by the introduction of a *pollster*—an agent that executes "cryptographic and data functions" on the voter's behalf (who must therefore be trusted). This in itself is a security issue, as is the authors' admission that, given two identical ballots, only one would be counted. Note further that although a voter can state that they wish to abstain, a misbehaving voter may deliberately not indicate this. Such misbehaviour again allows authorities to conspire to vote on that voter's behalf. Finally, as noted by Dini (2003), if the machine the voter uses crashes between contacting the validator and receiving a certificate back, the voter is never able to vote.

2.2.2 Mix Network-Based Protocols

Mix networks (Chaum, 1981) are a cryptographic primitive designed to simulate an anonymous channel between two endpoints, through a chain of proxy servers. They can be divided into two types: *decryption* mixes and *re-encryption* mixes. Decryption mix networks are the sort first proposed by Chaum, but both types work on a similar principle. In order to anonymously send a message m from Alice to Bob, Alice sends the message, encrypted in some way, to an *intermediate mix* proxy m_i. A number of other participants, each with their own messages for other destinations, do the same. When the mix proxy has received a sufficient number of messages, it forwards them in a random order to the next stage of the chain (which could be the intended destination, or another mix, depending on the desired level of anonymity). The way in which messages are handled at each mix in the *mix cascade* is where decryption and re-encryption mix networks differ. In a decryption mix, Alice must not only know the public key of her intended recipient, but also all of the public keys for each intermediate mix proxy. She first encrypts the message with Bob's public key, and then that, plus the destination of the message (Bob), with the mix which will receive the message before Bob does, then the one before that, and so on. In Chaum's work, a random seed r_i is added at each stage; this is not necessary for probabilistic encryption schemes:

$$\{r_n, \{r_{n-1}, \{...\{r_2, \{r_1, \{r_0, m\}_{\mathsf{Bob}}, \mathsf{Bob}\}_{\mathsf{mix}_0}\}_{\mathsf{mix}_1}...\}_{\mathsf{mix}_{n-2}}\}_{\mathsf{mix}_{n-1}}$$

Alice begins by forwarding the encrypted message to mix_{n-1}, who removes the first layer of encryption, giving

$$\{r_{n-1}, \{...\{r_2, \{r_1, \{r_0, m\}_{\mathsf{Bob}}, \mathsf{Bob}\}_{\mathsf{mix}_0}\}_{\mathsf{mix}_1}...\}_{\mathsf{mix}_{n-2}}\}$$

and then forwards the ciphertext to mix_{n-2}. The process continues until Bob receives and decrypts the message. Note that only one member of the mix cascade needs to be honest in order for Alice to remain anonymous, even if all other mixes collude. One of the main failings of decryption mixes, however, is that a single mix failure causes the message to be lost (as, at some stage, it will be not be realistically possible to decrypt it). Solutions to this problem involving a *threshold* mix using ElGamal threshold decryption have been presented (Jakobsson, 1998). The work required by the message originator is also proportional to the number of mixes.

The alternative *re-encryption mix networks* rely on public-key encryption schemes that permit re-encryption, such as ElGamal (discussed in Sections 3.1.1 and 3.1.4). Alice, the originator of the message m, does not need to know the number of mixes in the cascade, and only one encryption is required of Alice. She encrypts m with the public key of the mix itself, and forwards the ciphertext to the first mix server. This server takes a batch of input ciphertexts, re-encrypts them using some random seed, and forwards a random permutation of these re-encryptions to the next mix server. When the exit mix (the last mix server) receives the batch of ciphertexts, a quorum of cooperating mix servers can jointly decrypt the ciphertext, thence forwarding it to its destination. Re-encryption mix networks are clearly more robust, and require less work for the message originator. Much work has been done on further increasing this robustness (Jakobsson et al., 2002; Boneh and Golle, 2002; Holle et al., 2002; Golle et al., 2004; Wikström, 2005; Sako and Kilian, 1995) by making mix networks *verifiable*: i.e., able to prove that they are handling inputs in the correct manner.

Many election schemes assume the availability of an anonymous channel (including FOO, discussed earlier), implicitly meaning a mix network of some sort. Here, we discuss some of the most important protocols which use mix networks to ensure voting security.

2.2.2.1 Sako & Kilian

The Sako-Kilian protocol (Sako and Kilian, 1995) was the first to use ElGamal re-encryption to provide a provable, universally verifiable mixnet for electronic voting. Although the protocol does not assume a physical voting booth (an assumption which would rather hinder remote voting), it does assume the existence of a private, *physical* untappable channel, which is extremely hard to create.

The authors first explain a standard zero-knowledge bit commitment scheme, involving a prover committing to a value b by generating a pair (B, S_b), where B is a "blob", and is sent to the verifier. The prover can later apply a protocol open to B by sending S_b to the verifier, which allows the verifier to generate b using the two values B, S_b. The bit-commitment scheme used by Sako and Kilian is such that it is computationally infeasible to generate another S_b such that b can be obtained in any other way. A *chameleon* blob is one that allows the verifier, on input (B, b), to generate the correct S_b. The protocol uses chameleon blobs to allow the verifier to forge proofs.

Sako and Kilian's protocol details a universally verifiable mix network, whose security is ensured by forcing each mix server to prove that messages are being correctly processed. For brevity, and as it is not appropriate to the focus of this work, we do not discuss this here.

Protocol The protocol is summarised by the authors in four steps:

1. First, for each voter i, the final counting center posts encryptions of 1-votes and 0-votes (note that this protocol therefore only allows for elections with two possible outcomes, in its basic form). The center commits, using chameleon bit commitments, to the random ordering of these votes, and proves the pairs are correctly constructed. He opens the ordering to the voter (i.e., reveals the order by applying open to the chameleon commitment), through the aforementioned untappable channel

2. Each mix server shuffles the two votes for each voter, committing to that shuffle using chameleon commitments, and proving the correctness of every shuffle, again revealing this shuffle to the voter on the untappable channel

3. The voter, keeping track of the initial ordering and how the order was permuted by each mix, knows which vote is which, and can submit one of these votes

4. All votes are sent to the counter using a verifiable mix network, and thence tallied.

Sako and Kilian provide a more detailed set of implementation steps:

Constants	$p = kq + 1$ (p, q prime); $g = (g')^k \mod p$ (g' is a generator, mod p)
Centre j's public key	$\gamma_j = g^{x_j} \mod p$
Centre j's secret keys	x_j
Voter i's public key	$\alpha_i = g^{a_i}$
Voter i's secret key	a_i
1-vote	m_1
0-vote	m_2

The protocol proceeds as follows:

1. The last mix centre, n, executes the following for each voter i:

   - Commit a random bit string $\pi^{(i,n)}$ length $l + 1$ using public key α_i. Let $\pi_k^{(i,n)}$ denote the kth bit of this string.
   - Generate $v^0 = (\overline{G_n}, \overline{M_n}) = (g^{r_{2n}}, m_0.\overline{\gamma}^{r_{2n}})$. G and M represent two parts of the message sent to each mix; r is a random number, which is fresh for each message pair.
   - Generate $v^1 = (\overline{G'_n}, \overline{M'_n}) = (g^{r_{2n-1}}, m_1.\overline{\gamma}^{r_{2n-1}})$. In both of the above, $\overline{\gamma}$ represents $\prod \gamma_i$.
   - Place (v^0, v^1) if $\pi_1^{(i,n)} = 0$ and (v^1, v^0) otherwise. Prove that the placed pair is a combination of 1-vote and 0-vote in a similar technique to those described previously, l times (l is a security parameter)

2. The centre reveals to the voter which vote is the 1-vote, by decommitting $\pi^{(i,n)}$

3. The next centre, $n - 1$, execute the following with each voter i:

- Commit a random bit string $\pi^{(i,n-1)}$ length $l+1$ using public key α_i. Let $\pi_k^{(i,n-1)}$ denote the kth bit of this string.
- Generate $(\overline{G_{n-1}}, \overline{M_{n-1}}) = (\overline{G_n}.g^{r_{2(n-1)}}, \overline{M_n}.\overline{y}^{r_{2(n-1)}})$.
- Generate $(\overline{G'_{n-1}}, \overline{M'_{n-1}}) = (\overline{G_n}.g^{r_{2(n-1)-1}}, \overline{M'_n}.\overline{y}^{r_{2(n-1)-1}})$. $(\overline{G_n}, \overline{M_n})$ and $(\overline{G'_n}, \overline{M'_n})$ are votes for voter i sent from the previous mix centre.
- Centre $n-1$ places the votes in this order if $\pi_1^{(i,n-1)} = 0$, and reversed otherwise. He proves that the pair is a combination of 1- and 0-votes.

4. The centre reveals how he placed the votes by decommitting $\pi^{(i,n-1)}$.
5. Steps 3 and 4 are repeated for mix $n-2$, and on to the first mix centre.
6. The voter, who can compute which vote is a 1-vote and which is a 0-vote, submits the vote he wishes to make to the first centre, which routes it back to the last one (counting centre) through a universally verifiable mix channel
7. After the last centre reveals the permuted votes, anyone can compute the number of votes m_0 and m_1

(Sako and Kilian, 1995, pp. 400–01)

Despite the protocol's achievements, it does have problems. In the form presented here, it supports only two-way voting, and requires a considerable amount of work from the voter. Further, as noted by Michels and Horster (1996), a coercer must not collude with *any mix*, or else the tally is at risk of being incorrect. One might further bring the scalability of the protocol into question, and also its applicability to remote voting, given the strong requirement of an untappable channel.

2.2.2.2 Juels, Catalano and Jakobsson: Coercion-Resistant Electronic Elections

The work of Juels et al. (2005), known as the 'JCJ' protocol, is widely regarded as being seminal in the field of remote electronic voting, and has spawned a popular implementation, Civitas (Clarkson et al., 2008). Their scheme requires only an anonymous channel, and uses mix networks to permute votes and voter credentials, ensuring voter anonymity. Juels et al. are the first to

provide a stronger model of real-world attacks, which must be considered in any remote voting protocol:

- **Randomisation** A coercer tells a voter to submit random ballot material. Thus, although neither voter nor attacker learns the vote, the choice of the voter is nullified. The authors note that the protocol due to Hirt and Sako (2000) is susceptible to this attack.

- **Forced Abstention** The attacker forces the voter to refrain from voting

- **Simulation** The attacker forces the voter to divulge her private key after registration but before voting. Thus, the attacker can simulate the voter's actions

Preliminaries When the voter V_i casts her ballot, she identifies herself with a digital signature, or some interactive authentication protocol. At this time, the voter incorporates a concealed credential, an encryption of a secret σ, provided to her by a registrar. The tallying authority $\mathcal{T}$ performs a blind comparison between these credentials, and a list $\mathbf{L}$ of encrypted credentials which are published by a registrar $\mathcal{R}$, leading to verification without revealing the identity of the voter. This method means that a coerced voter can give the attacker a fake credential $\tilde{\sigma}$, without demonstrating that the credential is invalid.

The list of participants begins with a set of Registrars $\mathcal{R} = \{R_1, R_2, \ldots, R_{n_R}\}$ who issue keys and credentials. Further, a set of Talliers $\mathcal{T} = \{T_1, T_2, \ldots T_{n_T}\}$ who process and count votes, and a set of voters $\mathcal{V} = \{V_1, V_2, \ldots V_{n_V}\}$ where i is an identifier for voter V_i. A bulletin board $\mathcal{BB}$ is assumed as explained before, which voters can read only once the voting process is complete. The authors define a *candidate slate* C to be an ordered set $\{c_1, c_2, \ldots, c_{n_C}\}$, each of which is a potential voter choice. A candidate is identified by an index j. A vector X, such that x_j is the number of votes for j, is the tally.

The authors' protocol makes use of threshold cryptography (as described in Section 3.1.1) and a Plaintext Equivalence Test tool *PET*, which allows comparison of two distinct ciphertexts to determine the equality of their plaintexts, without revealing those decryptions. Finally, the

system uses a standard re-encryption mix network, and non-interactive zero knowledge (NIZK) proof technique. The protocol follows five stages.

Protocol **Setup** Keypairs are generated for the registrar and tallier, the public parts of which are distributed.

Registration Having proved eligibility to vote, V_i receives from $\mathcal{R}$ a random value $\sigma_i \in_U \mathcal{G}$, which represents the voter's credential ($\mathcal{G}$ is an algebraic group described in more depth by the authors). This might be generated in a distributed manner between several registrars. $\mathcal{R}$ then adds $S_i = E_{PK_\mathcal{T}}[\sigma_i]$, where E represents a polynomial-time ElGamal encryption (and D the reverse) to a voter roll, L. L is maintained on the bulletin board and signed by $\mathcal{R}$.

Slate Publication $\mathcal{R}$ publishes a candidate slate C containing unique identifiers for all the candidates, with an election identifier ϵ.

Voting To vote, a voter casts a ballot for c_j containing two ciphertexts—one on her choice c_j, and one on the credential σ_i. The intricacies of this vote are left to the paper. The voter includes an NIZK proof of knowledge for c_j and σ_i, amongst other data.

Tallying To tally the ballots in $\mathcal{BB}$, $\mathcal{T}$:

1. Checks the proofs of correctness for each ballot. Let A_1 and B_1 denote the list of ciphertexts on candidates and credentials respectively.

2. Performs PETs on all ciphertexts in B_1, removing ballots with the same value—this prevents double-voting. Let A'_1, B'_1 respectively denote the resulting ciphertexts.

3. Applies a mix network to A'_1, B'_1 using the same permutation scheme for both, giving A_2, B_2.

4. Applies the mix network to L, the voter roll, then compares each value of B_2 to the ciphertexts of L using PET. A_3 is the resulting vector of votes which were made by valid voters.

5. Publishes the decryptions of all ciphertexts of A_3, i.e., the final tally.

(Juels et al., 2005, p. 71–2)

It should be noted that a voter is able to deceive a coercer by merely providing an incorrect σ_i value at the time of voting. In this manner, the voter is confident her vote won't be counted, but the coercer may be convinced otherwise. She can then vote at another time correctly. Note, however, that JCJ (and Civitas, discussed below) cannot protect from a coercer who stands over the shoulder of the voter to observe her behaviour, as she must execute a protocol which generates fake credentials. This is something which we aim to address in our work.

The authors themselves note that the scheme is impractical for large-scale elections, as it has an overhead for tallying authorities which is quadratic in the number of voters (this is mainly due to the inefficiency of the PET). The PET (Jakobsson and Juels, 2000) is, in fact, the main cause for criticism of the JCJ protocol: it involves several rounds of pairwise blind comparisons between all ballots and all credentials, making the tallying portion of the protocol particularly inefficient, and questioning the practicality of the protocol on a large scale. Smith (2005) later proposed alterations which improve efficiency, but introduce undesirable properties (such as ballot collisions) and a security flaw, as discussed by Weber et al. (2007), who suggested modifications which are now also considered broken, as an attacker is able to determine whether a vote with a known credential is counted or not. Research into removing the quadratic complexity (with respect to the number of votes) of the scheme is ongoing (Clark and Hengartner, 2011; Spycher et al., 2011).

2.2.2.3 Civitas

Civitas (Clarkson et al., 2008) is a real-world implementation of JCJ, discussed above. The scheme is a direct implementation of the scheme, with a few exceptions: Civitas distributes registration trust between several tellers, allowing production of credential *shares*; it uses multiple "ballot boxes" rather than bulletin boards, and importantly, it gives concrete consideration to the scalability of the scheme. Like JCJ, Civitas has a number of trust requirements, some of which affect the real-world practicality of the scheme.

Foremost, users must trust their voting clients. As we discuss in Chapter 5, this is a strong

assumption, and not always a wise one: viruses could easily affect a user's vote without their knowledge. The "ballot boxes" must be—at least in part—trustworthy, in that they return all votes to the tabulation tellers.

Unfortunately, Civitas does not change the way in which credentials and votes are tallied using PETs, meaning the scheme is as inefficient as that which preceded it. The authors note that, even with division of the electoral into smaller, more manageable voting wards, tabulation and duplicate credential/vote elimination is still expensive (Clarkson et al., 2008, p. 362).

2.2.2.4 Helios

Helios (Adida, 2008) is a remote election scheme based on the premise that some elections "do not suffer from nearly the same coercion risk as high-stakes government elections" (Adida, 2008, p. 335). As such, it is a scheme designed for low-coercion environments, which focuses more strongly on election integrity than on coercion-resistance. The protocol is based on the earlier *Simple Verifiable Elections* by Benaloh (2006) (itself based on the Sako-Kilian mixnet, discussed above), which we will discuss first.

Benaloh's work begins with the idea that complex cryptographic voting protocols are often far too difficult for non-specialists to understand, and hence deliberately abstracts away cryptography from the protocol he suggests. His work uses threshold encryption (specifically of the ElGamal variety) to ensure voter privacy, as does much work since. Benaloh's work continues with a discussion of an interactive proof method used in ballot tallying in his protocol, by which any observer can verify that the election was conducted correctly. The aim is to prove that two sets of encrypted ballots consist of the same votes (i.e., one may be a reencryption of the other). We begin with a set $\mathcal{B}$ of encrypted ballots, with the aim of 'shuffling' these ballots blindly (Benaloh, 2006, p. 3):

1. Each ballot $B_i \in \mathcal{B}$ is re-encrypted randomly to form B'_i
2. The set of re-encrypted ballots $\{B'_1, B'_2, \ldots, B'_m\}$ is randomly permuted to give $\mathcal{B}'$
3. A collection of n additional sets of ballots $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$ is generated in the same manner

4. A set of n challenge bits $c_1, c_2, \ldots, c_n$ is generated

5. For each i where $c_i = 0$, $\mathcal{B}_i$ is shown to be equivalent to $\mathcal{B}$ by revealing the re-encryption and permutation data. For each i where $c_i = 1$, $\mathcal{B}_i$ is shown to be equivalent to the original re-encrypted set $\mathcal{B}'$ by composing the re-encryption and permutation data used to create $\mathcal{B}_i$ with that used to create $\mathcal{B}'$, and revealing the composition

This interactive proof method removes the possibility of proving equivalence of any $\mathcal{B}_i$ to both $\mathcal{B}$ *and* $\mathcal{B}'$. The scheme can be made non-interactive using a standard Fiat-Shamir heuristic (Fiat and Shamir, 1986). Now, with the original ballot set stripped of identifying information, any party can generate their own shuffle, which is accompanied by a proof that the result set is equivalent. The final, encrypted shuffling can be decrypted by a quorum, allowing tallying.

Casting an actual vote is discussed at a rather high level. Benaloh discusses an in-person voting scheme only, in which the vote-creation device produces an encrypted ballot (in the form of a magnetic card, with the encrypted vote also printed on the front) for the voter's choice. The voter then signs in and swipes the magnetic card through a reader which stores his encrypted vote with her name. The magnetic card is then used as the voter's receipt, with which she can later verify his vote was cast (but cannot prove how she voted). Benaloh goes on to discuss various auditing options, including the option to immediately decrypt any ballot before casting.

Of course, this scheme is open to problems. Foremost, listing of the voter's identity with an encrypted vote allows a coercer to see that a voter *has voted*, meaning forced-abstention attacks are possible. Further, as noted in later work by Benaloh (2007), many other coercion attacks are possible: a voter could be given an encrypted ballot in advance; *chain voting* is possible[1]; a single vote-buyer could generate several encrypted ballots and remove them for later coercion of vote-sellers. Benaloh provides a number of solutions to this problem—for a voting-booth scenario—in his 2007 work.

[1] Chain voting is where a vote-buyer obtains a blank ballot, completes (but doesn't submit) it, and leaves the polling station. He gives this to a vote-seller, with some form of remuneration, and requests that it is cast—he can verify this later. The vote-seller then returns later with another blank ballot for the buyer, allowing continuation of the chain.

Helios (Adida, 2008; Adida et al., 2009) is related to the protocol discussed above, first in that a voter can produce ballots before authenticating herself in any way, and is only identified at the point of voting. Note that this means that *anyone* is able to test the validity of the ballot creation protocol. As Helios is a remote voting protocol, ballot production is in the form of a hashed ciphertext, which can be *audited* by the voter, who can acquire the plaintext and randomness used in the encryption and hash. She can continue to generate and verify fresh ballots, or *seal* a ballot by discarding the randomness and plaintext. Only then does she authenticate and submit her vote. Note that the ballot is not signed before casting, and that the voter can see the hash of her vote before it is sealed. Both of these issues can give rise to coercion in more at-risk environments; Helios deliberately avoids considering this problem.

Helios, like Benaloh, assumes that encrypted cast votes are listed next to their voter's name on a bulletin board. Some proportion of voters and auditors must check the correctness of the board (though, as noted by Neff (2003), very little auditing is actually required to elicit high confidence in election results). Helios shuffles all encrypted ballots and proves shuffling after the election closes, as with Benaloh's protocol. Finally, it decrypts each ballot, provides a decryption proof for each, and tallies the election. It seems apparent that this method of decryption, proof and tallying would be ineffective for a large election (but perhaps these are not considered by Helios, for the same reasons discussed above).

Adida's closing comments reflect Helios' stance that coercion resistance is often "futile from the start", namely because with any voting scheme (remote or otherwise), the voter gains little assurance of the software running on the election server, or client. As noted by Adida, a possible solution to this lies in hardware-rooted attestation (trusted computing). Note that as they stand, neither Helios nor the protocol on which it was based are receipt-free.

2.2.3 Homomorphic Encryption-Based Protocols

Homomorphic encryption schemes are extremely common in electronic voting protocols, for a simple reason: they allow efficient re-encryption and threshold decryption, and permit tallying of an election without the decryption of any single vote. Homomorphic encryption is also

commonly used as a simpler method to ensure universal verifiability of the tally. We discuss the ElGamal threshold encryption scheme, as well as the consequences of its homomorphic nature, more in Section 3.1.1, and so only summarise it here.

Given a generator g for an appropriate cyclic group G of order q (where p, q are suitably large primes and q divides $(p-1)$), we select a private key s at random, a public key $h = g^s$, and a random $\alpha \in_R \{0, \ldots, q-1\}$. An encryption of a message m is then constructed as

$$(x, y) = (g^\alpha, h^\alpha \cdot m) = (g^\alpha, g^{\alpha s} \cdot m)$$

Note that the holder of s (or a quorum who share it) is the only person who can decrypt this value without the calculation of a discrete logarithm. Now, if we take two ciphertexts

$$(x_0, y_0) = (g^\alpha, g^{s\alpha} m_0) \quad (x_1, y_1) = (g^\beta, g^{s\beta} m_1)$$

the product of those ciphertexts, *viz.* $(X, Y) = \prod_{i=0}^{1} (x_i, y_i)$, is equal to

$$\begin{aligned}(X, Y) &= (g^\alpha \cdot g^\beta, g^{s\alpha} m_0 \cdot g^{s\beta} m_1) \\ &= (g^{\alpha+\beta}, g^{s(\alpha+\beta)}(m_1 \cdot m_2))\end{aligned}$$

i.e., the product of the encryption of m_1 and the encryption of m_2 is the encryption of $m_1 m_2$. It is for this reason that ElGamal is known as a *multiplicative homomorphic cryptosystem*. If we carefully design the format of m, we can create a protocol which allows votes to be tallied without revealing any single vote: for example, choose $m = g^{M^{i-1}}$. Then if i is the index of the candidate being selected, an encrypted vote for candidate 2 would be represented as $(x, y) = (g^\alpha, h^\alpha \cdot g^{M^1})$. When multiplied together, two votes for candidate 2 would equal $(x, y) = (g^\alpha, h^\alpha \cdot g^{2M^1})$. In this manner, the tally is built up as more votes are multiplied.

Note that many protocols which use homomorphic encryption still assume the availability of some anonymous channel (such as that provided by a mix). The reason for separation here is that the manner in which votes are accrued and tallied is different.

2.2.3.1 Benaloh and Tuinstra: Receipt Free Secret Ballot Elections

Benaloh and Tuinstra (1994) were among the first to use homomorphic encryption in an election protocol. They were also the first to develop and implement the notion of receipt-freeness. Their paper discusses an important point—voting booths have the interesting property that they *require* a voter's vote to stay secret—meaning that it is impossible for a voter to be coerced. They go on to discuss that many existing protocols at the time failed because they allowed a voter to take a receipt saying how they voted. While desirable for the voter, this leads to the elimination of the ability of the voter to deceive someone else about his vote. Hence the authors introduce the property of *receipt-freeness* (Benaloh and Tuinstra, 1994, p. 544): a voter is able to vote without fear of recrimination, as no receipt or other proof of how they voted is given back to them. It should be noted that the presence of a fixed voting booth, as the authors suggest, could be seen to somewhat trivialise the problem of uncoercibility—after all, if a voter is forced to vote from a set location where they can be monitored, then it is hard for them to be coerced. However, the authors dismiss this:

> Even if we assume the presence of voting booths, the task of conducting an election in which privacy is maintained, coercion is impossible, and yet the tally is publicly verifiable is difficult. [Current paper-based voting systems] offer no mechanism whereby voters can develop true confidence in the accuracy of the tally (Benaloh and Tuinstra, 1994, p. 545)

Despite the fact that this paper is almost 20 years old, this point is still accurate—in the UK, voters have *no way* whatsoever of guaranteeing that their vote is actually counted. Hence Benaloh and Tuinstra suggest that the following three properties should be satisfied:

- **Privacy** No participant other than a voter should be able to determine the value of the vote cast by that voter
- **Uncoercibility** No voter should be able to convince any other participant of the value of its vote
- **Correctness** Every participant should be convinced that the election *tally* accurately represents the "sum" of the votes cast

Model Benaloh and Tuinstra introduce a set of voters, where each voter has two processes, V_0 and V_1 (in this respect, the protocol is similar to earlier work (Benaloh and Yung, 1986)). The tally t of the election is the number of voters who ran the V_1 protocol. The tally is *correct* if $t_L \leq t \leq t_H$ where t_L is the number of V_1 executions and t_H is the total number of voters, less those who ran V_0 (Benaloh and Tuinstra, 1994). The authors assume the existence of private and public channels, and a random number beacon.

A voting system is defined as *correct* by Benaloh and Tuinstra if, with m voters, "specially designated output common to all participants who follow correct protocols gives a common correct tally" with probability at least $1 - \frac{m}{2^N}$ for some security parameter N. Further, the system is *private* if no dishonest protocols can permit any participant to distinguish between voters running V_0 and V_1, with probability at least $\frac{1}{2} + \frac{1}{2^N}$ (Benaloh and Tuinstra, 1994).

Protocol The authors' protocol uses only one tallying authority, which has the ability to decrypt all votes. A ballot is an ordered pair, in which a random order encrypted 0 and 1 are stored, created by the voting authority. A beacon is used to prove that this ballot is legitimate, as described in Benaloh and Yung's earlier protocol (1986, discussed above); the interactive proof provides a small amount of information regarding the random ordering of 0 and 1 in the ballot, allowing the voter to choose which half of the ballot to use for the vote.

We note the introduction of two encryption primitives: if z_1 is an encryption of x_1, and z_2 of x_2, then there are two functions $\otimes$ and $\oslash$, where $z_1 \otimes z_2$ is an encryption of $(x_1 + x_2)$, and $z_1 \oslash z_2$ is an encryption of $(x_1 - x_2)$. These are properties of homomorphic encryption as mentioned earlier (Benaloh and Tuinstra, 1994). The protocol follows seven steps, involving a voting authority, voter and randomness beacon:

1. **Authority** Encrypt $N+1$ (security parameter N) pairs (x_i, y_i) where $0 \leq i \leq N$, such that each $x_i \in E(0)$ and each $y_i \in E(1)$ where E is a polynomial-time public key encryption function. For each i, let $\alpha_i = \min\{x_i, y_i\}$ and $\beta_i = \max\{x_i, y_i\}$. Reveal the pairs (α_i, β_i).

2. **Voter** Enter the voting booth

3. **Authority** For each i in $0 \leq i \leq N$, let c_i be the decryption $D(\alpha_i)$ of α_i. Transmit, over a *private channel*, the c_i value to the Voter

4. **Beacon** Generate N bits b_i for $1 \leq i \leq N$; send them over a public channel

5. **Voter** Leave voting booth

6. **Authority** For all i such that $b_i = 0$, open (α_i, β_i) by revealing the decryptions D of each, together with a certificate pair $(D'(\alpha_i), D'(\beta_i))$. For all i where $b_i = 1$, *connect* (α_0, β_0) to (α_1, β_1) by revealing either $D'(\alpha_i \oslash \alpha_0)$ and $D'(\beta_i \oslash \beta_0)$ if $D(\alpha_i) = D(\alpha_0)$, or the pair $D'(\alpha_i \oslash \beta_0), D'(\beta_i \oslash \alpha_0)$ if $D(\alpha_i) = D(\beta_0)$ (i.e., if $D(\alpha_i \oslash \beta_0) = 0$)

7. **Voter** To cast a 0-vote, let $v = \alpha_0$ if $c_0 = 0$, else let $v = \beta_0$. To cast a 1-vote, let $v = \beta_0$ if $c_0 = 0$, else let $v = \alpha_0$. Send v over a public channel. The result of the election is the tally of V_0 versus V_1 executions.

It should be noted that this protocol actually only requires *one* interaction from the voter, in the form of transmission of one bit. The authors note that as no third party can distinguish a 0-vote from a 1-vote, uncoercibility is obtained. Their proof is quite long, so is omitted here for brevity.

Benaloh and Tuinstra discuss a couple of problems with their protocol, but others seem to arise. Firstly, a dishonest authority could disrupt the election. This could be avoided by using several authorities, which would all have to collude to disrupt the election and determine how any voter voted. However, a single failure would cause the whole protocol to fail. The other obvious problem is that, in the version of the protocol given here, the authority gains full knowledge of how every voter voted. This is, of course, unsatisfactory. This is avoided using several authorities.

There is a further problem that there is no provision whatsoever for a 1-out-of-L general election—i.e., selecting from a number of candidates. Further, the authority could, in the voting booth, "fail to offer a voter the information needed to complete the interaction" (Benaloh and Tuinstra, 1994, p. 550), while claiming that it had. A possible solution, the authors note, is to provide two receipts, one fake, for the voting process.

Of course, the most serious problem is that the protocol does nothing to protect against

coercion by a party having some control over the voting authorities—further, if we do not assume a voting booth, coercion is simple. As the authors note:

> …a physical separation of the voter from possible coercive agents is fundamental to *any* uncoercible election protocol (Benaloh and Tuinstra, 1994)

This quote can perhaps be misinterpreted. Though it seems to be correct that the voter and coercer should *at some point* be separate, this need not be for the entire election, and may only need to be during registration and during the casting of a single vote. This is the stance we adopt later.

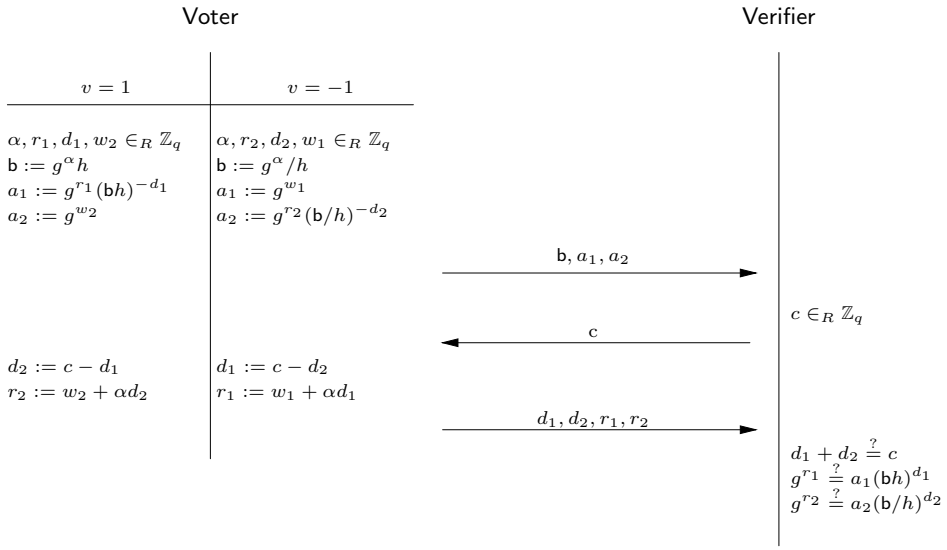
Hirt and Sako (2000) note that Benaloh and Tuinstra's protocol does not actually provide receipt-freeness—the voter is able, should they wish to, to generate a receipt for themselves, as a result of the properties of the mechanism used for multi-authority voting (the vote is distributed using secret-sharing). The reader is directed to the authors' paper (Hirt and Sako, 2000, p. 548) for a more in-depth explanation.

2.2.3.2 Cramer *et al.*: Multi-Authority Secret Ballot Elections

The work of Cramer et al. (1996) proposes a homomorphic scheme based on the principles of previous work by Benaloh and Yung, but using the discrete logarithm assumption. The protocol begins with a homomorphic encryption scheme—an extension of the work of Pedersen (1992)—that provides a proof of validity. The scheme is similar to a standard ElGamal encryption scheme. Encryption of a vote $v \in \mathbb{Z}_q$ proceeds with a random $\alpha \in_R \mathbb{Z}_q$ by computing $B = g^{\alpha}h^{v}$, for random, independent values of g, h. This B is later opened by revealing v and α. Note that for any two encryptions B_1, B_2 of values v_1, v_2, B_1B_2 is an encryption of $v_1 + v_2$, making the system homomorphic. The authors detail a zero-knowledge proof of validity for any ballot b (where $v \in \{1, -1\}$): this proof protocol is detailed in Figure 2.1.

Given the proof protocol, Cramer et al. (1996) detail the election protocol for a two-candidate election. We begin with n authorities $A_1, \ldots, A_n$, and m voters $V_1, \ldots, V_m$, where a quorum of $t < n$ authorities is required to reveal any single vote. Each voter then prepares a *masked vote* b_i as follows:

Figure 2.1 Encryption and Proof of Validity for a ballot b in Cramer et al. (1996)



1. The voter selects $b_i \in \{1, -1\}$ and calculates $B_i = g^{\alpha_i} h^{b_i}$ for some random α_i, and computes the proof as detailed in Figure 2.1. The voter then calculates

$$G_i(x) = \alpha_i + \alpha_{i1} x + \ldots + \alpha_{i,t-1} x^{t-1}$$

$$H_i(x) = b_i + \beta_{i1} x + \ldots + \beta_{i,t-1} x^{t-1}$$

   where $\alpha_{il}, \beta_{il}, l \leq t \in_R \mathbb{Z}_q$. The voter commits to these by calculating $B_{il} = g^{\alpha_{il}} h^{\beta_{il}}$.

2. The voter posts B_i, its proof of validity and all of the commitments to the bulletin board. All participants verify the correctness of B_i.

3. The voter sends each authority's share $(a_{ij}, b_{ij}) = (G_i(j), H_i(j))$ to each authority A_j using a private channel (the authors do not suggest whether the channel should be untappable).

4. A_j checks that his share is valid by checking that

$$g^{a_{ij}} h^{b_{ij}} = B_i \prod_{l=1}^{t-1} B_{il}^{j^l}.$$

5. To cast a vote, V_i posts $s_i \in \{1, -1\}$ to the bulletin board, such that $v_i = s_i b_i$ gives the desired vote.

Tallying of the election then involves all authorities participating:

1. Each A_j posts $S_j = \sum_{i=1}^{m} a_{ij} s_j$ and $T_j = \sum_{i=1}^{m} b_{ij} s_i$.

2. Each tallier checks each authority's (S_j, T_j) by checking that

$$g^{S_j} h^{T_j} = \prod_{i=1}^{m} \left(B_i \prod_{l=1}^{t-1} B_{il}^{j^l} \right)^{s_i}$$

and then from any t pairs (j, T_j) where (S_j, T_j) are correct, every tallier can compute the tally as

$$T = \sum_{j \in A} T_j \prod_{l \in A \setminus \{j\}} \frac{l}{l-j},$$

where A is any quorum set of t.

Note that the election scheme is compatible only with a two-way election. The authors suggest an extension to multiway elections, in which each voter simply produces several votes by running several instances of the protocol above in parallel (which is clearly not practical), or in which the ballot form is altered. It seems apparent that this scheme involves a considerable amount of work for the voter, even in the case of two-way elections—nevertheless, it is important as a grounding for future work by Cramer et al..

2.2.3.3 Cramer, Gennaro and Schoenmakers

The 'CGS' voting protocol Cramer et al. (1997) is an extension of earlier work (Cramer et al., 1996) for which the voter's work is linear with respect to a security parameter k. The work is more efficient for each participant by a factor of n (where n is the number of tallying authorities). As opposed to the scheme discussed above, in the CGS protocol, the voter need only submit a single ElGamal encryption of their ballot, plus a proof of its validity. The protocol, as usual, assumes the availability of a standard bulletin board (broadcast channel with memory), and a threshold ElGamal cryptosystem which uses Lagrange coefficients to reconstruct a secret, as in many protocols.

The paper notes that, since standard ElGamal is a multiplicative homomorphic encryption scheme, the product of any two ciphertexts c_1c_2 is actually the encryption of the product of their plaintexts, m_1m_2. In order to obtain an additive homomorphism (i.e., that the product of the ciphertexts is the encryption of $m_1 + m_2$), the authors alter their encryption operation so that a message m is now encrypted as $h^\alpha G^m$ (where G is a known generator of the group G_q used in the scheme), meaning that a product of ciphertexts would indeed be the encryption of $m_1 + m_2$. The downside of this change (which is similar to one we use in our work) is that a single discrete log calculation is required for decryption of the message, which is considered a hard operation. For small messages, however, the computation can be done efficiently, as noted by Cramer et al. (1997, p. 10).

The authors go in to introduce a *proof of validity*: namely, a proof that shows that any ciphertext is an encryption of either m_0 or m_1—where these are the two possible candidate selections for a ballot—without revealing which message is encrypted. If an encryption of a message is taken to be

$$(x, y) = (g^\alpha, h^\alpha m) \quad m \in \{m_0, m_1\}$$

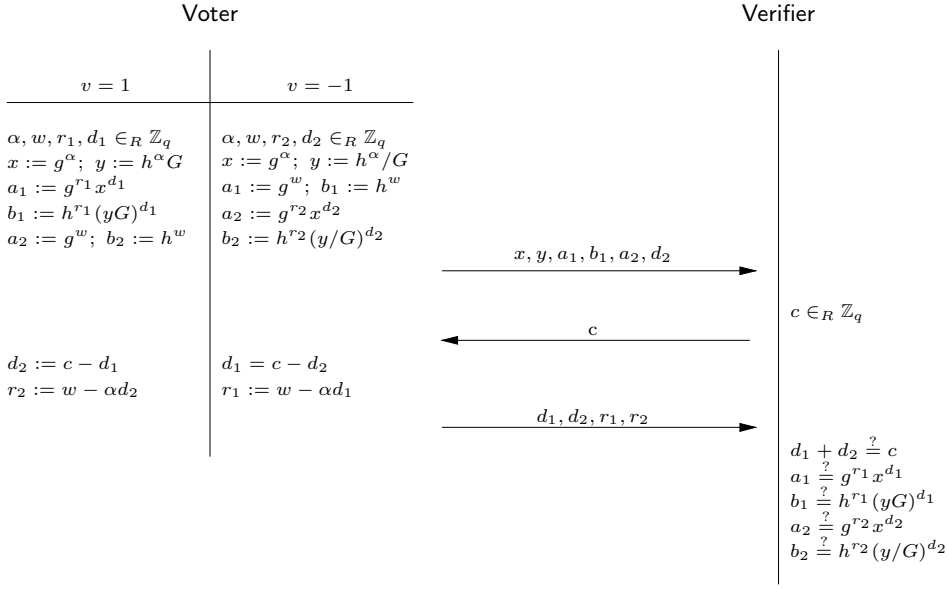
Then to show that (x, y) is a valid ballot, the prover must provide a witness-indistinguishable proof of knowledge of the relation given by

$$\log_g x = \log_h (y/m_0) \vee \log_g x = \log_h (y/m_1)$$

For a two-candidate election, the proof, and ballot, are formed in an interactive manner as demonstrated in Figure 2.2 (note that the method is similar to that of Cramer et al. (1996)). Note that this proof can be made non-interactive using the standard Fiat-Shamir heuristic: we extend the proof to a $k-$candidate election in Section 3.1.3, and use it in our work in Chapters 4 and 5. In the protocol in Figure 2.2, for a fixed generator G, message m_1 is represented as G, and m_0 as $\frac{1}{G}$. A ballot is then $(x, y) = (g^\alpha, h^\alpha G^b)$ for some $b \in_R \{1, -1\}$:

Given the protocol above, non-interactivity can be obtained by hashing a combination of the first message to the Verifier and some unique information (viz. the voter's identity) to give the

Figure 2.2 The CGS Proof of Validity for a two-party election ballot (x, y) (Cramer et al., 1997, p. 9)



challenge c. A voter casts the ballot by submitting the ballot (x, y), the proof above, and a value $e \in \{1, -1\}$, such that $v = be$ gives the required vote.

After voting is complete, each proof of validity is checked, and the authorities calculate the product

$$(X, Y) = \left(\prod_{i=1}^{l} x_i, \prod_{i=1}^{l} y_i \right)$$

from all ballots (x_i, y_i). The authorities jointly decrypt the product in the standard manner, giving $W = G^T$ as the result, where T is equal to the difference between yes and no votes. The value of T can be determined using $O(l)$ modular multiplications.

Note that the protocol is applicable only to a two-party scenario: however, the authors extend it to a k-way election by suggesting the selection of several generators G_i, one for each candidate, instead of only using one. This leaves a final tally $W = G_1^{T_1} \ldots G_K^{T_K}$, where the T_is are the final tally. The authors note that the computation of the necessary discrete log is still "feasible for reasonable values of l and K" (Cramer et al., 1997, p. 11)

This protocol has some particularly appealing features, and lends itself well to adaptation. We use some of those features in our own work, and note that the CGS scheme has been used for the basis of many protocols—one of which is due to Hirt and Sako (2000). It should be noted

that, as mentioned by Hirt and Sako, CGS is not a receipt-free protocol, but can be made receipt free with small modifications.

2.2.3.4 Hirt and Sako: Efficient Receipt-Free Voting

The scheme due to Hirt and Sako (2000) is strongly rooted in the work of Cramer et al. (1997). Indeed, the authors propose a 1-out-of-L (electoral candidates) voting scheme based on the CGS protocol. We begin again with a standard threshold ElGamal cryptosystem, modified such that a some γ such that $G = \langle\gamma\rangle : \gamma \neq g$ is used to form an encrypted vote $v \in \mathcal{V}$ as $(x, y) = (g^{\alpha}, h^{\alpha}\gamma^{v})$ (Hirt and Sako, 2000, p. 554). The authors note that for $L-$party elections, choosing the form of v is a challenge, drawing on the work of Cramer et al. (1996) to set $\mathcal{V} = \{1, M, M^2, \ldots, M^{L-1}\}$ where M is the maximum number of voters. Tallying again involves the computation of the discrete log of γ^T, having complexity $O(\sqrt{M}^{L-1})$ (Cramer et al., 1997).

The paper is one of the first, in our research, to use ElGamal reencryption to ensure ballot secrecy: namely, given a ciphertext (x, y), we can obtain a *reencryption* of the same plaintext (x_f, y_f) by selecting a value $\beta \in_R \mathbb{Z}_q$ and forming (xg^{β}, yh^{β}), where β is then a witness of the reencryption. Given this action, the paper also introduces two techniques: *the 1-out-of-L reencryption proof*, which proves that the reencryption of an encrypted vote e is contained in the list $e_1, \ldots, e_L$, and the *designated verifier reencryption proof*, which proves in a witness indistinguishable manner that an encrypted vote e' is a reencryption of another vote e.

1-out-of-L Re-encryption Proofs For a given vote (x, y), the authors wish to prove that there is a reencryption (x_f, y_f) of the vote in the list of all reencrypted votes $(x_1, y_1), \ldots, (x_L, y_L)$. The proof proceeds as follows:

1. The prover selects $d_1, \ldots, d_L$ and $r_1, \ldots, r_L$ at random, and calculates

$$a_i = \left(\frac{x_i}{x}\right)^{d_i} \cdot g^{r_i} \quad b_i = \left(\frac{y_i}{y}\right)^{d_i} \cdot h^{r_i} \quad \text{for } i = 1, \ldots, L$$

then sends these values to the verifier. These commit the prover to d_i, r_i for all i above, but

not for $i = f$: a_f, b_f commit the prover to $w = \beta d_f + r_f$, meaning that the prover can still change d_f and r_f after.

2. The verifier sends a challenge $c \in_R \mathbb{Z}_q$ to the prover, who modifies d_f such that $c = d_1 + \ldots + d_L$, modifies r_f such that $w = \beta d_f + r + f$ and sends all d_i, r_i (for $1 \leq i \leq L$) to the verifier.

3. The verifier now merely tests whether

$$
\begin{aligned}
c &= d_1 + \ldots + d_L \\
a_i &= \left(\frac{x_i}{x}\right)^{d_i} \cdot g^{r_i} \text{ for all } i \\
b_i &= \left(\frac{y_i}{y}\right)^{d_i} \cdot h^{r_i} \text{ for all } i
\end{aligned}
$$

Note that we can again use the Fiat-Shamir heuristic to convert the proof into a non-interactive one, where c is a hash of all a and b values with x, y and all x_i, y_i values.

Designated Verifier Re-encryption Proofs A designated verifier re-encryption proof is a technique to prove, only to a designated verifier, that some reencryption (x_f, y_f) is a reencryption of a specific ciphertext (x, y). The protocol uses the verifier's public key in proof construction (and thus secret key in verification), and is given in both interactive and non-interactive forms. As we use, and discuss, DVRPs in depth in Section 3.1.4, we do not cover them further here, but refer the reader to this section.

Protocol The protocol detailed in Hirt and Sako (2000) is rather high-level, but is similar in spirit to the CGS protocol. It assumes the existence of private, one-way untappable channels between authorities and voters (a particularly strong assumption, given the way in which this protocol uses those channels during voting). As long as a quorum of $t < n$ authorities remain honest throughout the protocol, it remains fair and private, and the protocol is receipt-free. Note that the paper assumes that the authorities are assumed not to collude with a coercer, unless the voter knows at least one honest authority.

The protocol begins with each possible vote being encrypted deterministically, with the ciphertexts being publicly known. The first authority shuffles the list by reencrypting its ciphertexts and permuting it, then submits it to the next authority, who does the same, and so on. Each authority provides (to the voter, via an untappable channel) a proof of how the list was reencrypted and permuted and (to the public) a proof of how the list was permuted only. Namely, the authority provides the voter with a DVRP for each of the reencrypted vote choices v in $\mathcal{V}$, and makes public a 1-out-of-L proof of re-encryption for every reencrypted vote choice.

Hence, at the end of the shuffling, the voter will be able to relate the list's original order to its current one, and thus pick the encryption representing her desired vote. Note that the private nature of the required untappable channel (even with respect to an in-person observer) means that no-one but the voter knows which ciphertext relates to which candidate, as the voter receives the specific permutation and re-encryption proofs used for her ballot. The voter then simply announces which of the encrypted votes e_i she wishes to cast.

The chosen encrypted vote is, presumably, added to the public bulletin board. Votes are added homomorphically in the style shown above to give the final tally. The paper does not suggest whether a random reencryption and permutation of each vote choice is done for *each* voter, or whether each voter receives the same permutation and re-encrypted set of votes. In the latter case, a security problem clearly exists; in the former, this seems like a remarkably expensive way in which to generate votes. The assumption of an untappable channel *during voting* also someone negates the possibility of the protocol being applicable to remote voting.

In later work, Hirt (2001) developed a related protocol which used a trusted "third-party randomiser" to minimise the shuffling work done by the authorities. Lee and Kim (2002) and Lee et al. (2004) extend this work, replacing the randomising party with a "Tamper-Resistant Randomiser" (smart card). Though offering an attractive solution, all of these ideas introduce a further layer of required trust, and those which use smart cards introduce a further layer of expense (particularly in the case of remote e-voting). Any attempt to drive the implementation of electronic voting should not create disincentives to implementation in this way.

2.2.3.5 'Paillier' Election Protocols

Those protocols using homomorphic encryption discussed so far exclusively use ElGamal encryption. However, alternative encryption systems, such as that of Paillier (1999) should also be considered. The Paillier cryptosystem offers a small advantage in efficiency, and has been adapted to a threshold version (Fouque et al., 2000). We note, however, that Paillier encryption-based schemes can often be *less* efficient than ElGamal-based schemes, as a larger security parameter is required for the same level of security (Hirt, 2010).

The Paillier cryptosystem begins with $n = pq$, a standard RSA modulus of two large prime numbers. Next, g is an integer of an order a multiple of $n \mod n^2$. The public key is (n, g) and the secret key is $\lambda(n)$, where $\lambda(n) = \text{lcm}((p-1)(q-1))$. Encryption of a message $m \in \mathbb{Z}_n$ involves random selection of some $x \in \mathbb{Z}_n^*$, and calculation of $c = g^m x^n \mod n^2$. Decryption is via the equation $m = \frac{L(c^{\lambda(n)} \mod n^2)}{L(g^{\lambda(n)} \mod n^2} \mod n$, where L, given the values $\{u < n^2 | u = 1 \mod n\}$, computes $L(u) = \frac{u-1}{n}$ (Paillier, 1999). Using threshold decryption requires the use of a distributed key generation algorithm among authorities to create a single public key, and shares of the private key. Each server runs a partial decryption of any ciphertext, generating a proof of the decryption, then forwards these shares to an entity which combines them (a concise version of the combination algorithm is given by Baudron et al. (2001)).

As we do not use the Paillier cryptosystem in our work, we detail only one exemplar protocol: that of Baudron et al. (2001). The protocol lists a number of entities: *voters*, *local authorities* (who collect a subset of all ballots, for a single local area), *regional authorities* (who receive all local authority results for a region), the *national authority* (who collects regional results), and a *trusted time stamp*, guaranteeing when a voter has voted. Again, a bulletin board is used, and the scheme uses a number of proofs of knowledge: that given an encrypted message $c = g^m x^n$, the prover convinces a verifier that he knows m; that the contents of an encryption can be proven to lie within a set of messages (i.e., a proof of validity); and that two encryptions can be proven to have the same plaintext (a PET).

Protocol The protocol begins with each authority publishing, on its own bulletin board, its public keys. The national authority uses a key pk, the regional authority pk_i, and the local authority $pk_{i,j}$ (envisage a tree structure). If ℓ is the number of voters, the authors define $M = 2^{\lceil \log_2 \ell \rceil}$.

A voter begins by selecting the three public keys of the national, regional and local authorities appropriate to him. He uses each public key to encrypt a value M^m, with m denoting his vote ($m \in \{1, \ldots, k\}$ for k candidates), giving three ciphertexts c_n, c_r, c_l. He generates proofs that each ciphertext contains a valid vote, then creates a PET which proves that each of the ciphertexts encrypt the same vote.

Tallying first involves the local authorities verifying all of the proofs, which have been posted to the bulletin board together with the voters' names and votes and signed by the time-stamp server. The local authorities then compute the product of the correct votes, and threshold-decrypt the tally for their own sub-tallies. These local tallies are published to the regional bulletin boards, along with the product of the elements on the local boards. The regional authorities are then able to calculate their own tallies (given the ciphertexts generated for them), and compare these with the local tally results, and the same for the national authority.

The protocol has several assumptions which must be satisfied. Foremost, in order to satisfy receipt-freeness and coercion resistance, the authors require a physical tamper-resistant device to hide the random data used during voting, or a secret, untappable channel between every voter and a 'randomiser' (Pointcheval, 2000).

Though the use of several levels of authority is an elegant way to reduce complexity, it seems that the redundancy created by each level checking the previous one's work somewhat mitigates any advantage. The protocol relies on a time-stamp server to prevent fairness being broken (i.e., to mitigate the issue that local authorities release partial tallies before the final tally is counted), and requires every voter to generate three votes, and three proofs (one of which is a PET, which is very expensive, as discussed in earlier sections).

Many other homomorphic encryption schemes exist in the literature (Kiayias and Yung, 2002, 2004; Schoenmakers, 1999; Damgård et al., 2010, for example); for brevity we do not address these further. The advantages of using a homomorphic cryptosystem are self-evident: computing

a product of all ballots is efficient and quick, and can be done by any observer of the bulletin board. Though (when using a modified form of ElGamal) tallying requires a discrete logarithm calculation, for small numbers of ballots this can be done quite efficiently (the baby-step-giant-step algorithm due to Shanks (1971) is an exemplar method). In our work, we use homomorphic encryption (in the form of the modified ElGamal scheme presented earlier) to form and tally ballots.

2.2.4 Paper-Based Protocols

The final class of e-voting protocol that we will discuss is *paper-based* protocols. These protocols, typified by *prêt-à-voter*, which we discuss first, generally involve the voter visiting a polling station and marking their ballot in some way on paper, submitting this, and electronically verifying that the ballot has been cast at a later time. One of the advantages of paper-based protocols is that they generally produce a paper 'receipt' with which the user (but no observer) can verify their vote later. This is proposed as an improvement on the current scheme used in many countries, where voters simply visit a polling station, cast a vote, and receive no receipt or means to verify that their vote was counted. Even with DRE systems currently in place, the voter "can only trust in the assurances of the manufacturers…that their vote will be [counted]" (Chaum et al., 2005).

Chaum (2004) first proposed a cryptographic paper voting protocol in which voters use a DRE machine, as in many existing electoral systems. In his scheme, however, the candidate selected by the voter also appears on a separate "printer" screen. Upon confirming a selection, the voter is allowed to select one of two rolls of paper from the printer, which acts as a receipt. The pattern printed on the paper is akin to an "unreadable and seemingly random pattern of tiny squares" (Chaum, 2004), not readable on its own, but able to create a readable image showing the voter's selection when superimposed upon the other, machine-retained receipt. This style of information encoding is known as *visual cryptography* (Naor and Shamir, 1995), a notion that has been built upon in later protocols (Chaum et al., 2007).

The voting machine keeps a copy of the receipt electronically, and then sends it to the election website some time later, deleting the half of the receipt left in the machine. In order to later check

that a vote was cast, one would enter the receipt's serial number on the website, checking that the receipt matches that being held by the voter. The reader is referred to Chaum's work for further information on tallying, as the techniques used (save for vote shuffling using mixes) are outside of the scope of this thesis.

2.2.4.1 Prêt-à-Voter

Prêt-à-Voter (Chaum et al., 2005) is a seminal protocol in the field of paper-based electronic voting protocols. The scheme uses a more regular representation of a vote than that of Chaum (above), with ballot papers that are printed in advance, and separable down the middle of the sheet. One clear advantage of this sort of scheme is that it is familiar to voters, and relatively simple to understand.

Setup and Voting We begin with a representation of the ballot form. Several ballot talliers are each allocated two keypairs, whose public parts are certified. The left half of a ballot form lists candidates, and the right half has boxes into which the voter makes her selection:

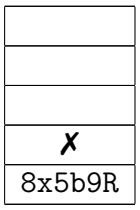
Figure 2.3 Ballot format in Prêt-à-Voter

Bob	
Rosie	
Frank	
Gemma	
	`8x5b9R`

The order in which the candidates are listed is not fixed: in fact, it is different on every ballot, and the permutation used is encrypted (multiple times) in the value `8x5b9R` shown in the figure. By way of an example, the authors discuss a simple election in which, rather than a random permutation, names are listed according to a cyclic shift of an initial order: if we assume that the original order is "Gemma, Bob, Rosie, Frank", then a shift of 1 gives the ordering "Bob, Rosie, Frank, Gemma", as above. The value (or *onion*) `8x5b9R` is then simply an encryption of '1' (note that in practice, the ordering is indeed a random permutation). The voter makes her selection

in the standard way, and then *detaches* the left half, which is destroyed. The right half reveals no information to any observer:

Figure 2.4 A separated, completed ballot in Prêt-à-Voter



This receipt is fed into a voting terminal, which sends the selected value (3 in this case, if we begin at 0) and the onion to the talliers, and returns it to the voter as a receipt. The talliers progressively decrypt the onion to eventually give the required shift from the original ordering (which even the voting terminal cannot know), allowing the original shift, 1, and the determination of the voter's vote for Gemma.

The Ballot Form We return to the way in which the ballot form (specifically, the onion) is generated. The authors begin by stating that the electoral authority generates a unique random seed sequence of $2k$ values (for k talliers), $g_0, g_1, g_2 \ldots g_{2k-1}$. A hash is then applied to each of the germs g_i, with the result being modulo v, for v candidates, giving $d_i : i = 0, 1, \ldots, 2k-1$. The offset for any ballot form is then

$$\theta = \sum_{i=0}^{2k-1} d_i \mod v.$$

It was mentioned earlier that each tallier has two keypairs. This is because each tallier performs two decryption mixes, one for each key. The onion is then formed of a nested encryption of each germ under each tallier's public keys, where tallier i has public keys pk_{2i} and pk_{2i+1}:

$$\text{Onion} = D_{2k} = \{g_{2k-1}, \{g_{2k-2}, \{\ldots, \{g_1, \{g_0, D_0\}_{pk_0}\}_{pk_1} \ldots\}_{pk_{2k-3}}\}_{pk_{2k-2}}\}_{pk_{2k-1}}$$

The talliers progressively decrypt the onion, beginning with the outermost two layers being removed by the first tallier. On the bulletin board, the first column shows the receipt as cast by the voter (allowing the voter to verify their vote has been cast). Tallier $k-1$ stores the position

of the cast vote as r, and then manipulates (r_i, D_i) for all i, where D_i is an ith level onion. The value r_{2k} is the position at which the voter original placed her mark (0-3, in the figures above).

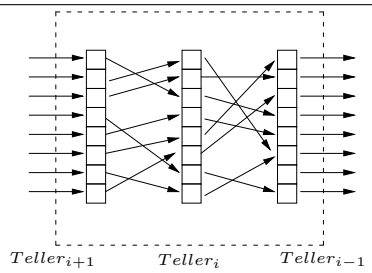
Talliers progressively accept a list of (r, D) values from the previous tallier, decrypting and shuffling twice and then passing along the new (r, D) list to the next tallier. So, for each (r_{2i}, D_{2i}) pair in the input, tallier $i-1$:

1. decrypts the first layer using his first secret key sk_{2i-1}, giving germ g_{2i-1} and onion D_{2i-1}
2. hashes the germ, and takes the result $\mod v$ to give $d_{2i-1} = \mathsf{hash}(g_{2i-1}) \mod v$
3. calculates $r_{2i-1} = r_{2i} - d_{2i-1} \mod v$
4. forms (r_{2i-1}, D_{2i-1})

The tallier now shuffles the list and posts the result to its middle column on the bulletin board. The tallier repeats the process using his other private key sk_{2i-2}, shuffles again and sends this new list to the next tallier, who repeats the whole process again.

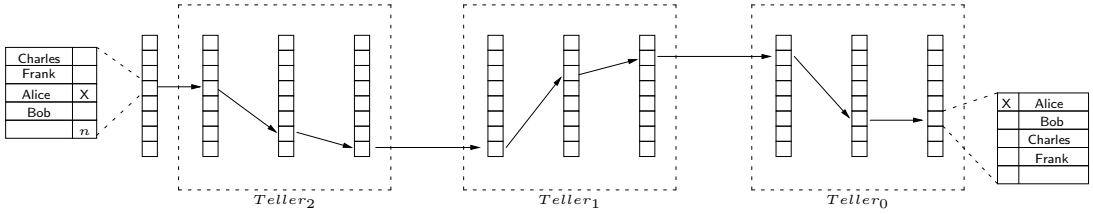
When all talliers have done this, the final output will be pairs (r_0, D_0) as in the Onion value above, giving the final votes in the original ordering. The actions of a single tallier are represented in Figure 2.5, and a single vote's progress through all talliers' mixes is shown in Figure 2.6.

Figure 2.5 A Single Tallier in Prêt-à-Voter (Chaum et al., 2005)



A useful property of the original Prêt-à-Voter scheme is that it is open to auditing by any interested party. Auditors are able to sample a random subset of all printed ballots to ensure that the onions are correctly formed, and, as detailed in the paper, concerned voters could in fact create 'dummy ballots', in which the voter's vote is sent to the talliers, the onion is decrypted, and the vote is returned to her for verification (of course, the vote is voided). This provides the

Figure 2.6 A Vote's Full Progression in Prêt-à-Voter (Chaum et al., 2005)



voter with assurance that her actual vote will be counted. Should the voter not wish to cast an actual vote, she could also simply have the ballot machine send off the onion, and return the shift value. As the authors note, however, all of the voter-auditing options are subject to attacks, if the ballot-printing authority colludes with any of the talliers.

It is also possible to audit the actions of each tallier, asking each tallier to reveal either the outgoing or incoming link for each of the (r, D) pairs in his list (random partial checking). We refer the reader to the original paper (Chaum et al., 2005) for more details here.

It should be noted that the protocol, though elegant and appealing (especially to to non-technical voters) has some issues. Some method must be used to ensure that voters who are permitted 'dummy votes' are not able to cast more than one ballot for real, and to ensure that chain voting is less of a risk. Some forced manner in which to destroy the left-hand-side of the ballot is essential (the authors address this in future work). The fact that the scheme uses a decryption mix network is also a point of failure: although privacy of the voter is assured if even one of the mixes is honest, *failure* of any mix means that no ballots encrypted whose onions are partially encrypted with its keys will be decryptable. We must also trust that the authority responsible for ballot creation can be trusted entirely. The authors claim that the protocol is "readily [adaptable] to remote voting" by distributing ballot forms by post. Naturally, in environments where coercion is a risk, this is not the case.

2.2.4.2 Prêt-à-Voter: Re-encryption Mixes

A later version of Prêt-à-Voter (Ryan and Schneider, 2006) introduces a number of fixes to the issues present in the original protocol. It uses ElGamal encryption rather than RSA, allowing the use of re-encryption mixes, rather than decryption mixes. The first fix to the original protocol

is that ballot forms are now generated by a set of l clerks, where each contributes to the seed, and all would need to collude to obtain the ordering of the candidates. We begin with a set of decryption talliers, holding shares of the private counterpart of a standard ElGamal public key (p, α, β_T) (the notation used in the paper; (q, g, h) in standard notation, with $G = \langle g \rangle$ known). These talliers, like those in the original scheme, decrypt the onion after voting. The authors also define a set of Registrars with private key shares for the public key (p, α, β_R), where the public key is used to construct each ballot.

Clerk C_0 generates a batch of seeds s_i^0 at random, and thence a batch of pairs of onions, by encrypting each s_i^0 such that an additive homomorphism is possible. For some γ generator of the appropriate group, then, C_0 generates

$$(\alpha^{x_i^0}, \beta_R^{x_i^0} \cdot \gamma^{-s_i^0}), (\alpha^{y_i^0}, \beta_T^{y_i^0} \cdot \gamma^{-s_i^0}),$$

for random values x_i^0, y_i^0. Each of the remaining clerks now re-encrypts each pair, permutes the list and transmits it to the next clerk. This gives two final onions, the *registrar* and *tallier* onions, where the s_i values in both should match. Note that only a collusion of all clerks could lead to the original seed values being revealed (but a threshold of registrars could cooperate to obtain the seed).

Ballot Creation The ballot itself can now be stored digitally, in an encrypted manner. Voters now receive a ballot of the form:

Figure 2.7 Empty Ballots in Prêt-à-Voter with Re-Encryption

$\texttt{onion}_L$	$\texttt{onion}_R$

A device in the voting booth reads the first onion, $\texttt{onion}_L$, and decrypts it (this may be done in a non-distributed manner, by storing the key in the machine, or in a threshold set of registrars).

The seed s is obtained from the onion, and it is used to derive the permutation of candidates to print on the left hand side of the ballot:

Figure 2.8 Ballot Ready for completion in Prêt-à-Voter with Re-Encryption

Rosie	
Gemma	✗
Frank	
Bob	
$\texttt{onion}_L$	$\texttt{onion}_R$

Now, providing the booth device does not see the right-hand onion, it will not be able to link the candidate list with the Tallier onion, which will be on the voter's receipt. Voting proceeds as before: the voter selects a candidate (Gemma, here); the left-hand strip is destroyed, and the voter casts their vote with an official observing, at a different machine. Again, the vote is stored as $(r, \texttt{onion}_R)$—where this time the onion is equal to $(\alpha^y, \beta_T^y \cdot \gamma^{-s_i})$. The authors note that it would also be feasible to generate another receipt, retained by electoral officials, to later check should problems arise (cf. the VVPAT system proposed by Mercuri (2002)).

Once votes are submitted (i.e., pairs (r, D)), an immediate issue is how to re-encrypt the pairs whilst providing privacy to the voter. As noted by Ryan and Schneider, re-encrypting only the onion is not sufficient. Hence, for a simple cyclic shift of ballot indexes, the authors suggest 'absorbing' r into the onion, viz. $(\alpha^y, \beta_T \cdot \gamma^{r-s_i})$: i.e., an encryption of the term $r - s_i$. Now, the r value is encoded into the onion, and so the onion can be re-encrypted alone by a standard re-encryption mix network. Once mixing is complete, a threshold of decryption talliers is then able to extract the plaintext values from the onion, giving several terms $\gamma^{r-s_i} \mod p$. The authors note that despite the apparent intractability of calculation of the discrete log of each of these terms, the manner in which they select the s values (from a binomial distribution) makes search of $\mathbb{Z}_p^*$ very efficient.

The way in which ballot forms are printed on demand, of course, removes the ability to audit them before the election (or, indeed, after). One solution is a two-sided ballot (Ryan and Schneider, 2006, p. 323), which, for brevity, we do not discuss here. The random partial

checking approach suggested for original Prêt-à-Voter is still appropriate, however, where the re-encryption seed is revealed by each mix.

Again, the issue of remote voting with Prêt-à-Voter is a complex one with respect to coercion. One of the authors' suggestions is akin to that of the credentials used in the JCJ protocol (Juels et al., 2005): a voter simply casts a credential (invalid if coerced) with his vote, where the validity of a credential is not apparent to an in-person observer. How voters *receive* these credentials is not discussed; nevertheless, this idea is similar to one which we use in our own work.

2.2.4.3 Derivatives of Prêt-à-Voter

Many extensions of the two Prêt-à-Voter protocols discussed above have been developed, and are still being developed today. In Ryan (2007), a version of Prêt-à-Voter is developed which presents a human-readable paper trail to the voter, marked with a ballot identifier number which is unlinked to the ballot. The scheme seems attractive, but, as noted by the paper, is subject to the same potential issues as all VVPAT-based schemes: coercion is more likely, and corruptions (possibly deliberate) of the paper audit count could lead to the digital count being questioned (even when correct). It seems that work required to mitigate these problems, as well as to set up the VVPAT in the first place, makes the scheme rather unrealistic.

Scratch & Vote (Adida and Rivest, 2006) uses Paillier encryption with Prêt-à-Voter-style ballots. Each ballot form is augmented with a 2D barcode containing an encryption of the candidate ordering, and "scratchable" surface protecting a plaintext ordering of the candidates on the ballot: in order to audit a ballot, the voter scratches the surface and verifies its contents publicly against the barcode value, voiding the ballot in the process. To cast a ballot, the scratch-strip is removed unscratched by an electoral officer, rendering the remaining ballot unlinkable to any candidate.

Prêt-à-Voter schemes have also been developed which support write-in ballots (Schneider et al., 2011)—long a problem for coercion-resistant elections, single transferable vote elections (Xia et al., 2007), and code voting (Ryan and Teague, 2009). This last version of the protocol involves sending a physical sheet of 'codes' to each voter (by postal mail, for example), where

each candidate has a random 'voting code', and the letter has a single 'acknowledgement code'. The voter then logs onto a website to cast his ballot, providing the ballot ID number from the letter with the appropriate voting code. The voting server then interacts with a quorum of trustees in order to return the acknowledgement code, which is verified against that on the letter. Note that this scheme is receipt-free, but also prevents the voter from seeing exactly how her vote was recorded (as opposed to traditional code voting, where every candidate has a specific acknowledgement code). Of course, the scheme can also not defend against an over-the-shoulder observer, except if the voter has the opportunity to vote once unobserved. The idea of 'securely' transmitting the code sheet to the voter, whilst also preventing the voter receiving multiple fake code sheets from an adversary (a postal office worker who deliberately intercepts her mail, say) is also questionable.

2.2.4.4 ThreeBallot

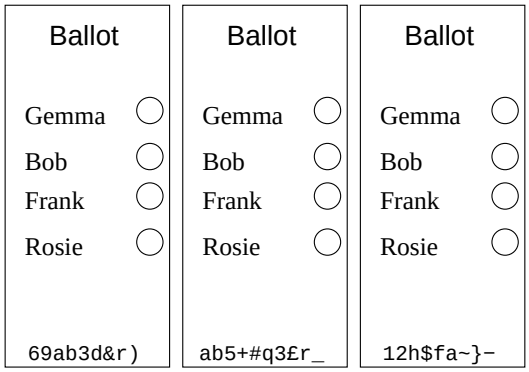
ThreeBallot (Rivest and Smith, 2007), and its closely related relatives VAV and Twin, were designed to provide the security guarantees of traditional cryptographic voting, but without cryptography. We will discuss ThreeBallot here.

The protocol, like Prêt-à-Voter and Punchscan (see Section 2.2.4.5), is from a class known as "End-to-End (E2E) Verifiable" protocols, which produce cryptographically secured ballots, ballot receipts, and auditability at every stage of the election, such that the results can be independently verified. Though ThreeBallot uses no cryptography, it is typically classed as E2E because of the properties it offers.

The protocol begins with a voter receiving a 'multi-ballot': three paper ballots on separate sheets (a voter might select these ballots from a bin full of empty ones). An example is given in Figure 2.9.

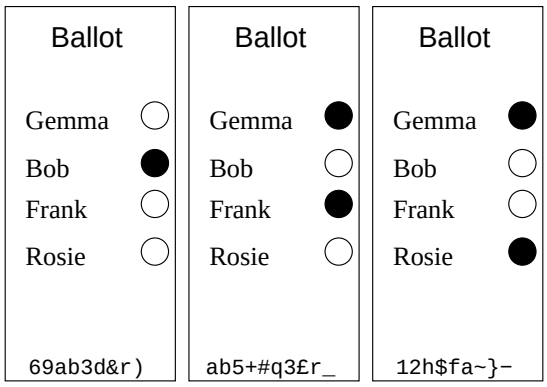
Each ballot is identical, but each has a unique (machine-readable) ballot ID on the bottom. To vote *for* a candidate, the voter fills in any *two* (but only two) of the 'bubbles' next to that candidate. To vote *against* a candidate, the voter fills in only *one* bubble for that candidate. If any candidate has zero or three bubbles filled in, the ballot is void. Note that the scheme (based on

Figure 2.9 A ThreeBallot multi-ballot (Rivest and Smith, 2007).



approval voting) allows approval of more than one candidate. In Figure 2.10, we vote only for Gemma.

Figure 2.10 A completed ThreeBallot multi-ballot, voting for Gemma.



The voter then submits her ballots to a 'checker' which validates her submissions, and allows her to select *one* of the ballots as a receipt. This ballot is reprinted onto different paper, and this copy is returned to the voter. All three ballots are dropped into a ballot box. It is at this point that a number of assumptions are made: the voter must "secretly and arbitrarily" select a ballot for a receipt, the machine must not remember which she chooses, and the voter must sign her name to indicate having voted (until which point, the vote must not be cast).

At the close of the election all ballots are posted on a bulletin board, and can be publicly verified. The voter can check for her receipt ballot in the list. The tally is such that each candidate's total is increased by the number of voters (i.e., even candidates with 0 votes have a tally of n for n voters).

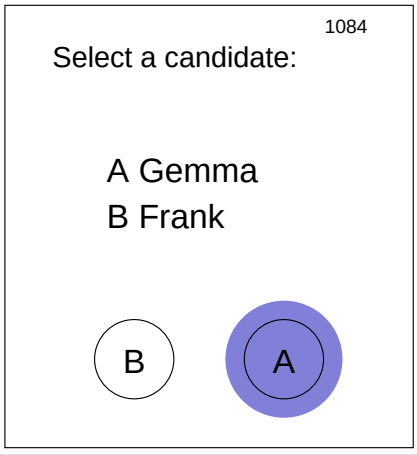
At first glance, ThreeBallot appears to be an attractive protocol. Unfortunately, it is subject to

several problems and attacks: from a usability perspective its demands upon the user are considered far too extensive (consider an election with more than 50 candidates, where the rules on how to vote apply!), and the system is entirely dependent on the reliability and trustworthiness of the 'checker' machine. Further, the protocol is subject to a number of security attacks. First, since the ballot ID is used to identify each ballot, the protocol implies that each voter only remembers the ID on the receipt she retains. This is a naïve assumption. More worrying is an attack in which an adversary with access to the ballot box can alter the outcome of the election in his favour—without detection—by buying receipts from voters with a filled bubble only for him, and altering ballots with single marks for other candidates to have single marks for him (Appel, 2006). These problems all cast something of a negative light on ThreeBallot, which is now seen as a non-practical protocol for large-scale elections.

2.2.4.5 Punchscan

The Punchscan system (Fisher et al., 2006; Popoveniuc and Hosp, 2010), originated by Chaum, is a paper-based voting system which claims not to rely on complex machinery of the sort used by Prêt-à-Voter and ThreeBallot. An exemplar Punchscan ballot is shown in Figure 2.11.

Figure 2.11 Completed ballot, voting for Gemma, in Punchscan



In the figure, the ballot consists of two pieces of paper. The top layer lists the candidate names and which letters they relate to, with the letters being randomly chosen. The bottom layer has the same letters, in random order, visible through holes in the top layer. To make a selection, a

voter makes a mark using a 'bingo'-style ink dauber, such that the diameter of the ink blot is larger than that of the hole, leaving ink on both sheets. One layer (committed to before the election) is destroyed; the other is submitted and scanned as the vote, and is returned to the voter as a receipt. The scanned ballot is uploaded to a website which the voter can visit. Note that neither layer reveals the vote on its own.

To tally the vote, the system uses a *Punchboard* to determine the order of the candidates on the ballot. The Punchboard is a set of three tables, the *permute*, *decrypt* and *result* tables. The permute table stores the identity of the ballot, the ordering of the top and bottom layers, commitments to these, and the mark that was made by the voter. In the decrypt table, the mark made on the ballot is translated into the actual vote, according to any differences in ordering between the top and bottom layers. Rows from the permute table are mixed (permuted) before entering the decrypt table, and permuted again afterwards, making the whole process rather like the action of a mix network. The result table simply stores the final vote from each row.

Note that if the Punchboard were to be released to the public, any vote could be linked back to its voter; however, if the board were kept secret, votes could be altered by an adversary in the decrypt table (Fisher et al., 2006). As a compromise, the full board, encrypted, is released, and parts of it are audited at random, to make significant changes noticeable.

The Punchboard is audited before the election, by selecting half of the ballots listed on it and decrypting (spoiling) the ballot information on those rows. This information is made public, making the integrity of the Punchboard universally verifiable (with high probability). When the election is complete, auditors select and decrypt either the left or right half of the decrypt table, revealing either transformations from original ballots to their intermediate decryptions, or transformations from the intermediate to the end vote (thus not revealing any one path from encrypted ballot to vote).

The authors note a number of trust requirements: foremost, the device used to print the ballots must be trusted to do so according to the permute table. The software present in the device used to process votes at each polling station must run trusted software. Popoveniuc and Hosp (2010) provide a number of security proofs for the manner in which the auditability of

Punchscan makes its integrity unquestionable, and the manner in which it satisfies voter privacy (anonymity). The protocol is clearly receipt-free (and E2E verifiable), and despite small issues with its usability in deployment (Essex et al., 2007), the system seems promising. Again, one must question how we would apply remote voting to Punchscan, and further whether revocable anonymity could work in any remote implementation.

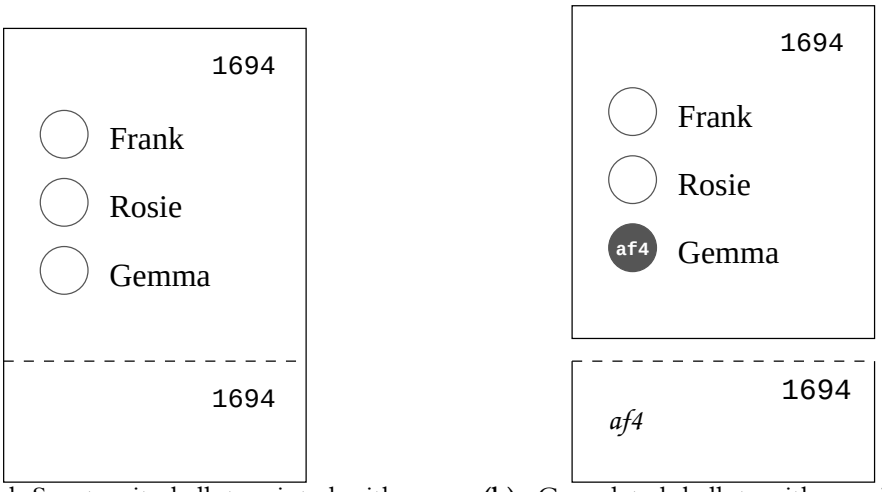
2.2.4.6 Scantegrity I and II

Scantegrity (Chaum et al., 2008b) were originally designed as enhancements for standard optical scan voting systems (with the idea that voters do not change their method of voting), minimising any increase in cost at polling stations already using optical scan. Scantegrity is in fact much like Punchscan, on which it is based: the main difference is that the ballot sheet itself does not change, with the exception of a bar-coded serial number being printed on a perforated corner of the ballot ('chit'), which the voter uses together with the letter relating to her selected candidate in order to verify her vote. The way in which dispute resolution relies on the trustworthiness of the polling station officer to not link a voter to a vote is, however, questionable, and an attacker can force the voter to produce a pre-specified receipt.

The protocol was quickly superseded by Scantegrity II (Chaum et al., 2008a), which is what we will discuss here. The protocol does not rely on paper chits, or on election officials in the case of dispute resolution. It requires no extra polling station equipment, apart from a small change to how ballots are printed. However, the system unconventionally introduces *invisible ink* and *decoder pens* to the voting process. Ballots are designed as shown in Figure 2.12a.

A ballot with a detachable base, each part tagged with the same identity number, are prepared. Each candidate's *bubble* has a unique random sequence of characters (a confirmation code) written inside it in *invisible ink*: hence, when the voter first sees the ballot, none of the codes are visible. The voter marks her choice using a special 'decoder' pen, which makes the confirmation code visible—see Figure 2.12b. An optical scanner detects the dark mark made by the pen, but *does not interpret the confirmation code*. If the voter wishes to later verify that her vote was counted, she

Figure 2.12 Scantegrity II ballots



(a) Blank Scantegrity ballot, printed with invisible ink

(b) Completed ballot with receipt detached

writes the confirmation code on the bottom portion (her receipt), taking it with her. Note that the confirmation codes for candidates *not* selected *must* remain secret.

Before the election, a quorum of authorities share a seed for a random number generator, then generating enough pseudorandom confirmation codes for all candidates on all ballot papers. These values are entered into a table **P**, in the order generated by the random number generator. **P** contains a row for every ballot, linking confirmation codes to candidates for each ballot by its ID number, and hence is kept secret throughout. Another table, **Q**, contains each ballot's confirmation codes in rows (one row per ballot), with the order of those codes permuted, and their candidate links removed. The values in **Q** are committed to, and those commitments published. A table **R** contains a row for every confirmation code from **Q**, with a flag to denote whether a vote is present for that row, and link between the row and column in table **Q** (viz "row 2, column 1"), and a row and column in table **S**. This final table simply contains a column for every candidate, where every cell is a flag, set to true if a vote is present. Commitments to **Q** and **R** are published, and the full table **S** is published.

The voter is able to select two ballots when voting, deliberately voiding one for use as an audit ballot. The ballot used for voting has its receipt stamped by an election official, and the audit ballot receipt is stamped as voided. After voting is complete, values are entered into **R** according to the opened values in **Q**. Election officials, like in Punchscan, audit the process by

either opening the **Q**-pointer, or the **S**-pointer, in **R**, for every row of the table, at random. For spoiled ballots, officials open both of these pointers, as well as the commitments. The standard assumptions about a small number of audited ballots leading to high election integrity guarantees apply here: we refer the reader to Chaum et al. (2008a) for more details, and diagrams of the tallying process.

Scantegrity and Scantegrity II are more examples of interesting paper-based, E2E systems which seem very promising for 'polling station', in-person elections. Again, however, Scantegrity II seems to make some rather strong assumptions: foremost, to us, is the notion of "invisible ink". It seems hard to believe that a reliable system for printing ink which could both not be seen *at all* by the naked eye, and also become immediately visible when drawn over with certain inks, could be developed. Indeed, as the authors note, any such ink would change the reflectivity of the surface on which it was used. The use of such technology naturally makes systems such as this inappropriate for remote voting, but the techniques used to achieve verifiability are relevant nonetheless.

2.2.5 What is Wrong with e-Voting?

Given the breadth of work which has now been covered, we consider an obvious question: with so many electronic voting protocols and systems, what is wrong with electronic voting? Take-up has been extremely poor, with many countries deciding against an implementation of electronic voting in the near future, and some organisations in the UK staunchly opposed to it (Open Rights Group, 2007).

Randell and Ryan (2005) suggest that the ultimate goal of electronic voting is to "develop an e-voting scheme that is both secure and sufficiently understandable to gain as high a level of public trust as is achieved by a number of existing manual voting schemes, such as that in current use in the UK" (Randell and Ryan, 2005). This is a very important point—as has already been discussed, no voting system will be successful unless it is trustworthy *and trustable* by the general public. As Cranor suggests, "simultaneously achieving security and privacy in electronic polls is a problem that must be solved if the Internet is to be used for serious large-scale…elections"(Cranor, 1996).

To some extent, these problems have been solved, but not while satisfying other problems in electronic voting, such as the balance between remote elections and coercion resistance.

2.2.5.1 Current Problems

As Jorba et al. (2003) discuss, "The wealth of problems in current electronic voting systems has led to a shake in public confidence in electronic voting in general" (Jorba et al., 2003). The authors refer to the fiasco surrounding the aforementioned US Presidential elections, but one can clearly see that problems surround both paper and electronic voting. Postal voting in the UK is wide open to fraud or coercion, and it is desirable to trace those who deliberately break the law in these situations—revocable anonymity is an ideal solution.

Unfortunately, attempts at large-scale electronic voting have thus far been prone to failure. The most serious example is that of the US Presidential elections in Florida, in 2000. The equipment concerned was found to be faulty, ballots were often confusing; there were mistakes in the actual recording of votes, and problems with missing ballots. All of these failures apparently led to up to 6 million votes being lost (Jorba et al., 2003). As Chaum et al. (2005) note when discussing the American DRE system, "With…[the] DRE,…the voter at best gets some form of acknowledgement of the way they cast their vote. After that, they can only hope that their vote will be accurately included in the final tally" (Chaum et al., 2005). It is hardly surprising that the public have lost trust in such systems.

The problems with the US elections in 2000 are further described by Mercuri (2002). She notes that currently, the public will be unwilling to adopt a system unless it is sufficiently close to what was available before. Specifically, she suggests a human-readable paper trail (VVPAT) with every vote, noting that there must be "a way to backtrack vote totals from actual ballots that came from…legitimate voters"(Mercuri, 2002, p. 46). Protocols such as that of Ryan (2007) provide some form of paper trail. We question the relevance of a paper trail to remote, Internet-based voting protocols, however: indeed, Paul and Tanenbaum (2009) suggest that a return to using paper ballots is arguably "a return to the problems that prompted the use of electronic machines" in the first place. Although paper ballots have a definite place in modern elections, we believe

that a system allowing users to vote over the Internet is critical to higher turnout. We posit that, as suggested by Bochsler (2010), though remote, Internet voting may not dramatically increase turnout by disaffected voters immediately, it will increase participation by those voters that are engaged, but merely unwilling to participate by traditional means. This done, with time, we feel that turnout using a remote election system would substantially increase.

Volkamer and Krimmer (2006) note that, due to the large number of security requirements of Internet (electronic) voting which have yet to be satisfied, a large-scale application of electronic voting is still a long way away (Volkamer and Krimmer, 2006). Indeed, Weber et al. (2007) and Liaw (2004) state that before any successful electronic election system can be implemented, a plethora of problems, including vote selling and coercion, have to be addressed. Until these problems are solved, along with satisfaction of the trustworthiness requirement for public use, electronic voting will not be successful. It seems somewhat odd, on this note, that several systems have been designed which appear to satisfy the requirements of electronic voting, but none have been used on a large scale. This is perhaps due to a combination of complacency in the current paper systems, unwillingness to adopt a new scheme considering the cost associated with it, and a lack of confidence in cryptographic security.

If governments are to adopt any remote election system, perhaps the most prevalent problem is the complexity of existing protocols. Many of those protocols discussed above have a high level of complexity to the user. That complexity is perhaps necessary in order to satisfy the demanding security requirements that voters have, and it can possibly be mitigated by front end software which "hides" much of the cryptography. Whether this in itself is acceptable to the security community is an important point, but the satisfaction of the end user is more important: *many* end-users are likely to be satisfied with a solution which appears user-friendly and offers a guarantee of their vote being counted. In fact, it is the satisfaction of the user which is most pivotal to the success of electronic voting:

> Many individuals expressed concerns over the security and privacy of e-voting and felt that substantial reassurance would need to be offered by the government prior to implementation. Establishing and maintaining public confidence in the security and privacy of the electoral system appears to fundamental [sic] in achieving legitimacy for e-voting (Local Government Association, 2002, p. 5)

Some users, naturally, will require a stronger guarantee that their privacy is ensured, and their vote is counted as cast:

> It is not difficult to imagine a simple, stand-alone device which offers a rich user interface but has no responsibilities beyond providing the voter with an encrypted ballot…how then are voters to gain confidence that these devices are accurately creating encrypted ballots which reflect their intentions? (Benaloh, 2006, p. 2)

Until we can satisfy the stringent requirements of the most cautious end users, electronic voting is likely to remain unsuccessful.

2.3 Anonymity and Revocable Anonymity

Part of the aim of this thesis is to consider the application of *revocable* anonymity to electronic voting. This is a notion which has not been considered before, despite receiving considerable attention in other computer security fields. In this section, we will briefly consider what is meant by 'anonymity' and revocable anonymity, particularly with regard to electronic voting, and how these requirements might be realised, considering their implementations in related fields.

From the perspective of digital transactions, anonymity can be defined as the ability to hide a user's identity, under *all circumstances*, for any transactions. Pointcheval (2000) splits anonymity into two parts:

- **Unlinkability**: It should not be possible to link two transactions made by the same user
- **Untraceability**: It should not be possible to match any transaction to a given user

It should be noted, of course, that a protocol providing untraceability alone is not sufficient. If a user's transactions are linkable, then it is possible to trace them. Pointcheval goes on to split anonymity into *strong* and *weak*, where it is, respectively:

- Not possible to guess a link between a transaction and customer (except with negligible probability), or
- Possible to guess or know such a link, but not to prove it (except with negligible probability)

Pointcheval (2000, p. 6)

Further, Guan et al. (2002) discuss anonymity as being categorised by who is anonymised: *sender* anonymity protects the identity of the sender (which is what one would prefer for electronic voting); *receiver* anonymity protects the receiver, and *mutual* anonymity protects both. This distinction is important when considering any system providing anonymity. Naturally, in an electronic voting protocol, we are interested only in sender anonymity.

Anonymity is essential in many digital protocols. Marx (1999) discusses several reasons for providing anonymity in a generic sense, many of which can be applied to electronic voting:

1. To facilitate the flow of information—for example, anonymity crime report lines
2. To obtain personal data for research—for those that would not usually give such information away
3. To encourage attention to the content of a message—i.e., to encourage a counter to look at the vote, rather than the voter
4. To obtain a resource or encourage action involving illegality—this is a negative issue, but nevertheless, as von Solms and Naccache (1992) confirm, a valid reason for anonymity
5. To protect strategic economic interests
6. To protect one's time, space, and person—e.g., having an ex-directory telephone number
7. To aid judgements based on specified criteria—e.g., permitting the vote of an ex-criminal in an unbiased manner
8. To protect reputation
9. To avoid persecution based on one's actions
10. 'Traditional expectations'—the author suggests the invention of the caller ID system in the US, but this can be applied to voting. Voters generally *expect* their vote to be anonymous.

Marx (1999, p. 102)

It is apparent that having a system which is completely anonymous is often dangerous. In the case of electronic commerce, this could lead to the 'perfect crime', in which an adversary can blackmail a user to make a transaction, without fear of recrimination (von Solms and Naccache, 1992). However, in other fields, a completely anonymous system is also a dangerous thing. In distributed computing, one might use anonymity to prevent other users (possibly competitors) from knowing anything about a job submitted to a computing grid: as Ciaraldi et al. (2000) note, the lack of identifiability in a distributed transaction can mean that "neither the distributor of a computation nor the client in a computation can know with assurance with whom they are communicating"(Ciaraldi et al., 2000, p. 195). They go on to note that knowing a distributor's identity is often crucial to assure the safety of a distributed process. Further, knowing the identity of a client may be crucial, because of the potential trustworthiness of the clients (Ciaraldi et al., 2000).

In systems providing anonymous access to the Internet, full anonymity can again be a dangerous thing:

> On the one hand, users are concerned about their privacy, and desire to anonymously access the network. On the other hand, some organizations are concerned about how this anonymous access might be abused (Claessens et al., 2003)

> …anonymity can facilitate socially unacceptable or even criminal activities because of the difficulties in holding users accountable (Marx, 1999)

It is for these reasons that some sort of *revocable* anonymity, or partial *identifiability*, are proposed for all digital protocols, and in this case, electronic voting. Again, Marx gives several reasons why one might need identifiability:

1. Accountability—it must be possible to punish someone who commits a crime. If this threat of punishment is not there, then those originally not likely to commit a crime may be tempted to do so.

2. To judge reputation—one might determine the validity of someone's vote based on some identifying characteristic about them[1]

3. To aid efficiency and improve service

4. To guarantee interactions that are distanced or mediated by time or space—in the case of voting, one may wish to guarantee that one's vote has been counted. This justifies the use of a receipt, but receipts unfortunately open a voting system to coercion. Hence a balance has to be drawn.

(Marx, 1999)

It is clear that some balance needs to be drawn. It could be argued that all digital protocols could benefit from anonymity. However, those same protocols need some mechanism to trace and punish those who commit a crime. In the case of electronic voting, one might wish to punish a voter deliberately attempting to vote twice. To some degree, though, the reasons *for* revocable anonymity in electronic voting almost do not matter: the British electoral system requires that a voter can be linked to their ballot, and hence a solution which does this is required.

Having satisfied that revocable anonymity is useful, it can also be a bad thing. Davida et al. (1997), discussing revocable anonymity in digital cash, notes that *blackmailing* is a possible problem. Consider motoring: the steering wheel lock has reduced unattended car theft, but has increased car-jacking, where the car is entered (and the driver removed) while it is being driven. Similarly, if a system is available by which a voter can prove they have been blackmailed, the criminal may have to kill the voter to prevent his being caught. This seems rather drastic for a protocol in which the most a coercer can gain is a single vote per instance of blackmailing, but the example is still valid.

Defining the Degrees of Anonymity It is appropriate to consider the *degree* to which a user of an electronic protocol remains anonymous. Shields and Levine (2000) defines several levels of anonymity between an attacker and user, based on the work of Reiter and Rubin (1999):

1. *Provably Exposed*: The attacker can prove that Alice is the initiator of a transaction

[1] Admittedly, this reason contrasts with reason 7 justifying anonymity!

2. *Exposed*: The attacker is convinced that Alice is the initiator, but there is some possibility that she is not.

3. *Probable Innocence*: Alice appears no more likely to be the initiator than to not be, but appears to be more likely than all other entities

4. *Beyond Suspicion*: Alice appears to be no more likely to be the initiator than any other entity

5. *Absolute Privacy*: The attacker cannot determine the presence of any communication

(Shields and Levine, 2000, p. 35)

Clearly, when trying to enforce a secret ballot election, we need a protocol which either provides beyond-suspicion-anonymity or absolute privacy for Alice. This requirement is tempered by the issue that Alice may need to be linked back to her ballot: as a result, there must be some route by which Alice can be unquestionably identified, but *only* by a trusted, authorised entity.

2.3.1 Approaches to Remote Anonymity

We now briefly discuss approaches to the provision of (revocable) anonymity in various fields in computer security. We begin by addressing two general pieces of work, and then focus on electronic commerce (as we discuss later, electronic voting can be seen as a type of electronic commerce).

2.3.1.1 Self-Scrambling Anonymizers

Pointcheval (2000) introduces two concepts in his paper, *anonymity providers* and *revocation centers*, which are analogous to trusted third parties. Anonymity providers "certify re-encrypted data after having been convinced of the validity of the content, but without knowing anything about the latter" (Pointcheval, 2000). In the field of digital cash—which we discuss in Section 2.3.1.3—this might guarantee that a coin is that of a certified legal user. The identity of this user is protected until revoked by a revocation center. Pointcheval raises the important point that anonymity providers could *charge* for their service; clearly, in electronic voting this would not be appropriate,

but in many communications, users may not *want* to pay for anonymity, as they feel it isn't necessary.

Pointcheval's paper, which is written with a view to digital cash, discusses the problem of double spending, which originated as a result of blind signatures:

> …a crucial problem came from over-spending, which refers to the situation in which a user spends the same coin two or more times. An inherent quality of digital data is that perfect copies are easy to make; therefore such fraud cannot be avoided, but just detected in the best case. Then…detection is done later. However, what may be done if the coin in completely anonymous? (Pointcheval, 2000, pp. 1–2)

This is a very important point, which is analogous to the situation in electronic voting. A voter could in theory vote twice, if they have been given some digital token with which to do so. Hence it is necessary to revoke their anonymity. Pointcheval discusses a later work of Chaum et al. (1990), which uses *cut-and-choose*: the user's identity is stored in the coin in such a way that spending the coin twice automatically reveals the identity of the user. Such a solution could be used in electronic voting; however it should be noted that this is an inefficient idea.

Pointcheval's scheme uses the idea of *designated verifier undeniable signatures*, including an undeniable proof scheme as follows:

- A key generation algorithm $\mathcal{K}$ which outputs random secret and private keys
- A proof algorithm $\mathcal{P}(\mathsf{sk}, m)$ which, on an input m, outputs an 'undeniable' signature s on m. This proof does not on its own convince anyone; one has to interact with the owner of sk.
- A confirmation algorithm $\mathsf{Confirmation}(\mathsf{sk}, \mathsf{pk}, m, s)$, between the signer and verifier, where the signer proves validity of the pair (m, s) to the verifier
- A disavowal algorithm $\mathsf{Disavowal}(\mathsf{sk}, \mathsf{pk}, m, s)$ where the signer tries to prove that (m, s) is *not* a valid pair

(Pointcheval, 2000, p. 4)

It should be noted that, except with negligible probability, it is not possible to succeed in both Confirmation and Disavowal.

As mentioned, Pointcheval's protocol is in the context of digital cash. It begins with *withdrawal*; a revocably anonymous coin is a certified message which includes the user's public key. In this case, the coin may be the user's public key pk_{Alice} encrypted with the revocation center's public key: $c = \{pk_{\text{Alice}}, r\}_{RC}$, where r is a random value. The user then signs a message containing this encryption, her public key and r: $\sigma = sign_{\text{Alice}}(pk_{\text{Alice}}, r, c)$ and sends this to the bank.

Of course, the bank is now able to see everything about Alice, and hence verifies that the signature σ is valid, then returns a certificate pair (Cert_c, c) to Alice. Alice then uses a *self-scrambling anonymizer* to transform her ciphertext c into something different, thereby protecting her identity. She re-encrypts her coin $c = \{pk_{\text{Alice}}, r\}_{RC}$ into $c' = \{pk_{\text{Alice}}, r + t\}_{RC}$, which is possible via the ElGamal encryption scheme used. Next, she provides an undeniable signature s, using c as a public key associated with secret key (sk_{Alice}, r), stating the equivalence of c and c'.

This new c', along with c, Cert_c and the signature proving equivalence are sent to the *anonymity provider*, which is convinced that:

- c was converted to c' by Alice
- c is equivalent to c'
- Alice won't later be able to deny the relation between c and c' (because of s)

The anonymity provider will then return a new certificate $\text{Cert}_{c'}$, guaranteeing the authenticity of the coin.

Alice then spends a coin simply by proving that she owns it, by providing a signature demonstrating knowledge of the secret key (sk_{Alice}, r) (Pointcheval, 2000, pp. 6–9). As a property of the way coins are formed, double-spending (or two attempts at anonymising the same coin) will result in Alice's public key being revealed by a simple decryption.

The reader is directed to the appropriate paper for a more in-depth discussion of how this protocol works; it is summarised here for brevity. However, the idea is a good one—in voting, a user could obtain a token allowing her to vote, and then transform it into one which prevents her identity being disclosed (except by a revocation center, if she attempts to vote twice, for example).

2.3.1.2 Pseudonymity

The first author to suggest the use of pseudonymity was Chaum (1985), whose digital payment system suggested the use of several different *pseudonyms* for each user, where a pseudonym is simply some transformation of a user's identity[1]. Importantly, as Lysyanskaya et al. (2000) note, "each organization may know a user by a different pseudonym, or *nym*. These nyms are *unlinkable*: two organizations cannot combine their databases to build up a dossier on the user" (Lysyanskaya et al., 2000, p. 242). Further, a user can demonstrate possession of a credential to an organisation without revealing anything about the credential to that party. The authors proceed to give examples of several pseudonym systems, but note a common problem: "there it little to motivate or prevent a user from sharing his pseudonyms or credentials with other users"(Lysyanskaya et al., 2000), as in the example of an online magazine subscription.

Hence, Lysyanskaya et al. (2000) introduce the concept of *all-or-nothing* sharing. Each user has a *master public key*; the user should desire to keep the corresponding secret key secret. For example, the key may be registered with an authority as that user's master signing key, so that anyone having possession of the key would be able to forge signatures. Hence, the authors' system has the property that if a Alice shares a credential with Bob, she automatically shares her master secret key with him.

In the paper, the authors propose a certification authority, which simply enables a user to prove that his pseudonym corresponds to some master public key, whose owner can only share the credential by sharing his master secret key. Firstly, a user registers with a CA, giving his master public key and true identity, and demonstrating knowledge of the corresponding private key in some way. The user's nym with the CA is hence his public key. The CA returns a credential stating that the user is valid.

A user with such a credential is then able to *register* with an organisation. He does this in a secure interactive way. Both parties engage in a protocol NG, where both submit the public key

[1] Of course, in the simplest case, a pseudonym can be a completely different value, whose correspondence with a user's identity is stored at some trusted third party.

pk_O of the organisation, and each respectively submit their public and private master keys[1]. The protocol outputs a nym N for the user with organisation O.

When the user, U, later wishes to communicate with an O, he supplies a proof of knowledge of $(pk_U, sk_U, SI^U_{U,O})$, where $SI^U_{U,O}$ is secret information supplied to the user when requesting a nym via NG for O. In order for the user to gain a credential for U to interact with O, both undergo a *credential issue* procedure, to which the user submits his nym N and public and secret keys, along with the public key of the organisation and $SI^U_{U,O}$. The organisation inputs their public and secret keys, and the secret information $SI^U_{N,O}$ it received as part of the NG protocol. The user will receive a credential $C_{U,O}$ for use with that organisation, and the organisation will receive some private information relating to that credential.

2.3.1.3 Digital Cash and Electronic Commerce

We now discuss the approach taken in *digital cash* to revocable anonymity. We find this particularly relevant, as we believe that electronic voting can be classified simply as a type of electronic commerce. We clarify by way of an example. In electronic commerce, a user is issued a number of *coins* (payment tokens). These coins either have intrinsic value, being fiat currency, or effect a transfer of funds from one account to another when spent. Either way, they are usually encoded in some way with the user's identity. A user is allowed to *spend* a coin only once, where the act of spending is backed by a fundamental principle of digital cash: your payment should be untraceable to you, and multiple payments should be unlinkable to each other. However, if you commit a crime with your digital cash, authorised entities should be able to trace you.

Electronic voting can be seen as a version of electronic commerce with stronger requirements. Here, each user (voter) is given a *single* 'coin', which equates to that user's right to vote. When the user votes, they spend the 'coin', negating their right to vote again[2]. After this point, the result of the transaction is that the voter's vote is counted (rather like a service provided by the spending of a coin in electronic commerce). Of course, there are many differences between

[1] The authors mention a 'secure anonymous channel'—it is assumed that this is possible via a mix, or one of the other mechanisms described earlier.

[2] We note that a user may be permitted to 'double-spend' here, with the caveat that only their last transaction (vote) is counted.

electronic commerce and voting, but many similarities too, which at least merit the discussion of approaches to anonymity provision in digital cash.

Digital cash is a field that has not received a lot of attention in recent years: at the time of its inception, it could essentially have been described as a solution to a problem that customers didn't realise existed: how to make secure, anonymous payments.

Users of physical cash are rightly happy with their method of payment—it is secure, anonymous, and ubiquitous. However, remote payment is generally done by credit card. Again customers are happy with doing this, but perhaps without good reason. As Chaum et al. (1990) discussed several years ago, digital payment by credit card remains an act of faith for many:

> Each part is vulnerable to fraud by the others, and the [credit] cardholder in particular has no protection against surveillance (Chaum et al., 1990, p. 319)

Despite the article being written nearly twenty years ago, this point remains valid. The very nature of credit and debit cards allows all transactions to be traced, and it is very easy to defraud a payee, using a fake or 'cloned' card. There are further problems:

> Despite their widespread use and market penetration, [credit cards] have a number of significant limitations …, including lack of security, lack of anonymity, inability to reach all audiences due to credit requirements, large overhead with respect to payments, and the related inefficiency in processing small payment amounts (Jakobsson et al., 1999, p. 43)

In fact, the final problem mentioned above—the making of small payments, whether in person or over the Internet—is one that has *still* not been satisfactorily solved: credit card transaction fees mean that many businesses are unwilling to accept credit card payments under a certain threshold value. RFID small-value payment systems like PayWave™[1] and PayPass™[2] have mitigated these issues to some degree.

Despite these problems, the public are still willing to use credit cards for electronic payment. Thankfully, at least in Britain, in-roads have been made against these problems, through the use of Chip and PIN (which has not been as successful as hoped against fraud, as discussed by

[1] http://www.visa.co.uk/en/products/contactless.aspx
[2] http://www.paypass.com/

Anderson et al. (2005)) and the RFID systems mentioned above, for small amounts of money (Card Technology Today, 2007). Nevertheless, neither of these ideas provide any anonymity for the user, and arguably a legitimate user should have the right to make their transactions untraceable:

> From an ethical…perspective, giving someone the ability to conduct payments should not go hand-in-hand with knowing their whereabouts, their spending patterns, and their personal preferences (Jakobsson et al., 1999, p. 46)

This lack of anonymity is not only a problem from a personal perspective. When a customer makes a payment, the merchant has full access to their card details. With these details, a dishonest merchant could clone the customer's card. This problem is clearly exacerbated by Internet payment, where not only the merchant, but anyone with the ability to intercept a poorly encrypted payment protocol, has access to the user's data.

Hence current methods of electronic payment require a high level of *trust*, between the customer and merchant (bidirectionally), and between the bank and all parties. Of course, trust is needed in any digital protocol, but providing anonymity to a customer at least gives them some safeguard against fraud.

Early work by Chaum (1982) proposed the first 'digital cash' system, using blind signatures to effect unconditional anonymity for the spender. Whilst the protocol was better at providing anonymity for Alice than anything currently used, it is actually *too good* at making Alice anonymous. In fact, once she has obtained a coin from the bank, there is *no way* for anyone to trace her, on an application level. This is unacceptable—she is both subject to blackmail herself, and able to perform fraud by double-spending coins, for example. We have already discussed how total anonymity can lead to the 'perfect crime' (Pointcheval, 2000).

Jakobsson and Yung: Revokable Versatile Electronic Money The work of Jakobsson and Yung (1996) seems particularly apt in its work on anonymity for the user. The authors rightly suggest that it is not always appropriate to provide full anonymity, and it is further dangerous to trust one entity with anonymising a user's identity, thus meaning that two or more should be used:

> In order to allow funds to be traced, frozen and revoked, absolute anonymity has to be excluded. But we do not wish tracing to be possible at the whims of the bank, as transaction analysis can be used or sold for direct marketing…Therefore, we will split the tracing function between the bank and a consumer rights organization, the *ombudsman* (Jakobsson and Yung, 1996)

The ombudsman works in collaboration with the Bank to revoke anonymity, only when provided with sufficient evidence to do so by a Judge. As a result, if a coin includes a user's identity, it must be encrypted and signed with both the ombudsman and bank keys. This dual signature verification scheme means that it is neither possible for any party to obtain a customer's identity without co-operation of both parties, nor is it possible for any single party to mint money.

Like most digital cash protocols, the Jakobsson and Yung protocol is split into three phases: *withdrawal*, *spending* and *deposit*. Since the focus of this thesis is not on digital cash protocols, we do not provide a full description of the protocol: instead, we refer the interested reader to diagrams representing each phase of it, in Figures 2.13, 2.14 and 2.15.

For us, the relevant part of the protocol is in how Alice's identity is protected, and how her anonymity is revoked. As such, we briefly summarise these aspects of the protocol. In the *withdrawal* phase, where Alice's coins are minted, she selects session keys—K_B for communication with the bank, Bank, and K_O for the ombudsman, Ombudsman. She encrypts each session key with its counterpart's public key, and doubly encrypts her identity using the public key of the bank, and session key with the ombudsman. All session keys, and the symmetric encryption of the public part of the coin γ and double-encryption of her identity, are sent to the bank.

The bank verifies Alice's identity, creates a session number and sends the ombudsman's session key, symmetric encryption of γ and the double-encryption of Alice's identity to the ombudsman, who can derive his session key, the encryption of Alice's identity under the bank's public key, and a signature on γ. The process by which the coin is then joint-generated with the bank is irrelevant for us. The key point is that, once the withdrawal is complete, the bank stores the transaction number, Alice's identity, and the double-encryption of her identity; the ombudsman stores the transaction number, session key, γ and Alice's identity encrypted for the bank. To spend a coin, Alice sends γ with her blindly signed coin s (see Figure 2.13) to Bob. Given this information at a later point, the ombudsman and bank can cooperate to obtain Alice's identity (see Figure 2.15)

(Jakobsson and Yung, 1996). This rudimentary method of requiring multiple layers of encryption, and therefore the agreement of several parties, to protect Alice's anonymity, is important to our work.

Figure 2.13 Coin Withdrawal in Jakobsson and Yung (1996)

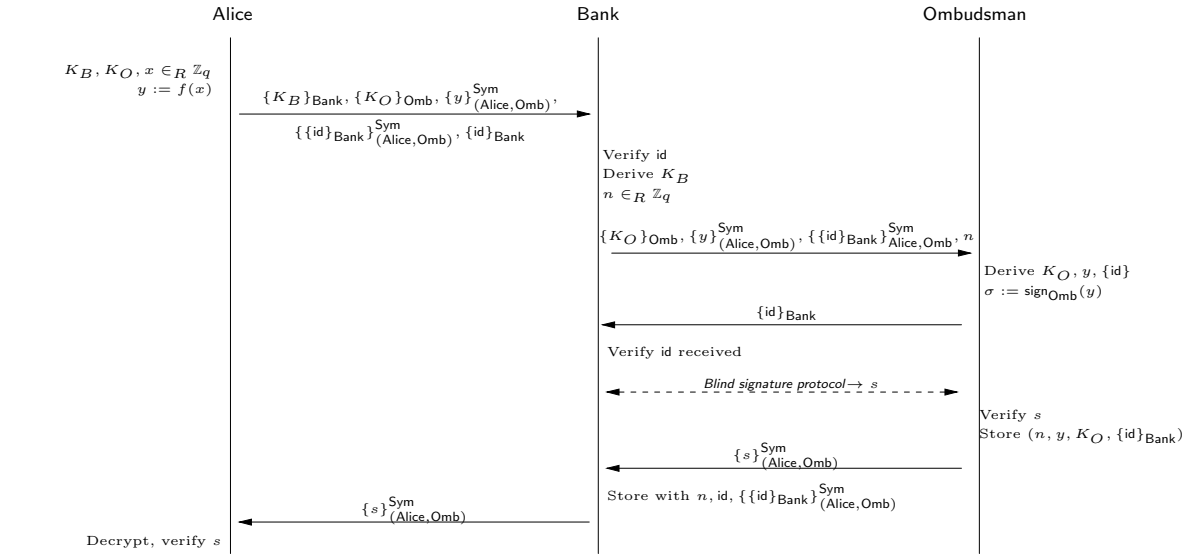
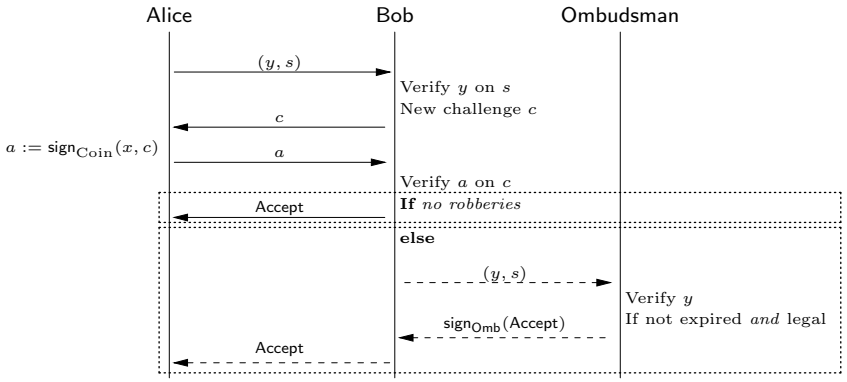
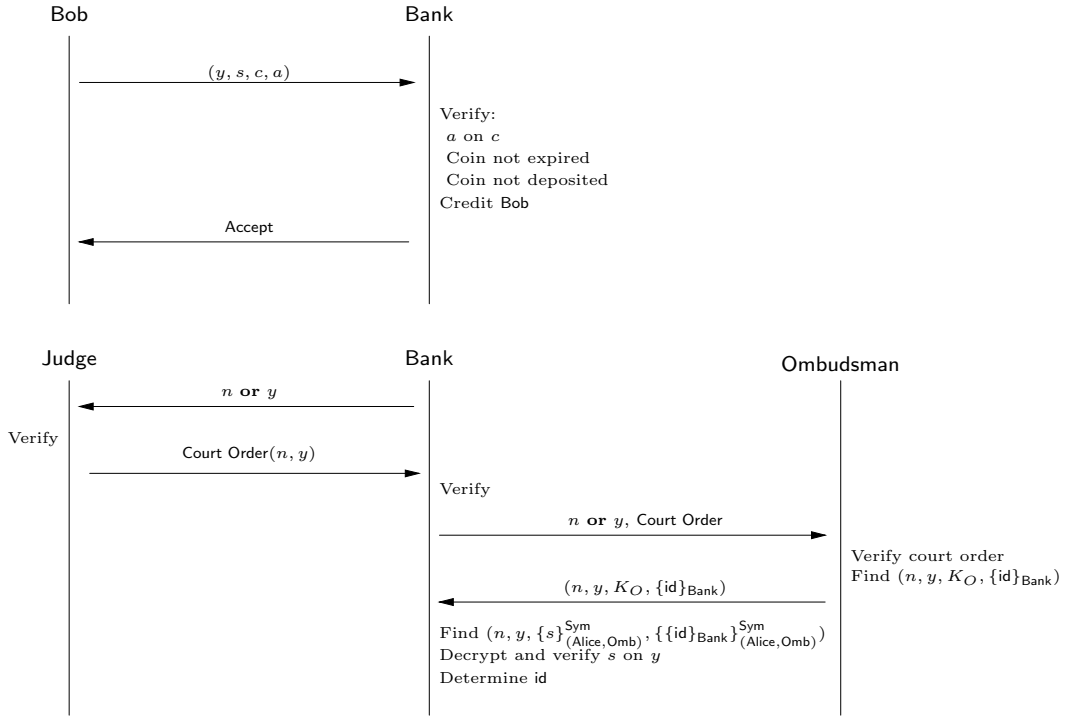


Figure 2.14 Coin Spending in Jakobsson and Yung (1996)



Kügler and Vogt: Auditable Tracing So far, we have made the case for revocable anonymity in a variety of protocols, including electronic voting. One of the most pressing problems with revocable anonymity, however, is the *fair tracing problem*: "no-one is able to control the legal usage of tracing, leading to the possibility of *illegal tracing*" (Kügler and Vogt, 2002, p. 137). To some degree, if we can trust that only a certain entity will have the key(s) required to revoke anonymity, then we can be assured that privacy is protected. However, what if this is not the case? What if a

Figure 2.15 Coin Deposit and Anonymity Revocation in Jakobsson and Yung (1996)



voter's anonymity is revoked with no clear reason, or by an unauthorised entity? Kügler and Vogt (2003, 2001, 2002) present a protocol which does not *prevent* this "illegal tracing", but instead all anonymity revocation *detectable* by the traced person. They term this notion "optimistic fair tracing". The theory behind this idea is that the threat of being detected deanonymising users is enough of a disincentive to rogue authorities to prevent it happening.

Here, we focus specifically on Kügler and Vogt (2003), a protocol designed for tracing revocation in digital cash. Though the protocol itself is not directly usable by us, many of the ideas presented in it are. Indeed, we discuss the idea of auditable anonymity revocation extensively in Chapter 6.

The protocol is based around the notion that when a customer 'withdraws funds' from the bank, essentially minting coins, she does so by encrypting her identity with some *tracing key*. In many protocols, there is a single tracing key for all coins and customers. In this protocol, a different public tracing key is issued for every customer, where these keys have a limited validity and must regularly be renewed. This allows two types of tracing:

- **Coin tracing**, where the withdrawn coins of a customer are deanonymised, so the bank recognises them when they are spent, and
- **Owner tracing**, where the coins deposited by a suspicious merchant are deanonymised, identifying their withdrawer (for example, tracing someone who makes a purchase from a known drug dealer).

Both of these are possible in the protocol, provided that the bank knows the private tracing key for the customer, and merchant, respectively. Whether or not the bank *does* know this key depends on the *key generation parameter*, a notion which will be defined later. Once a tracing key has expired, the bank reveals the parameter, enabling the customer or merchant to see if they were traced.

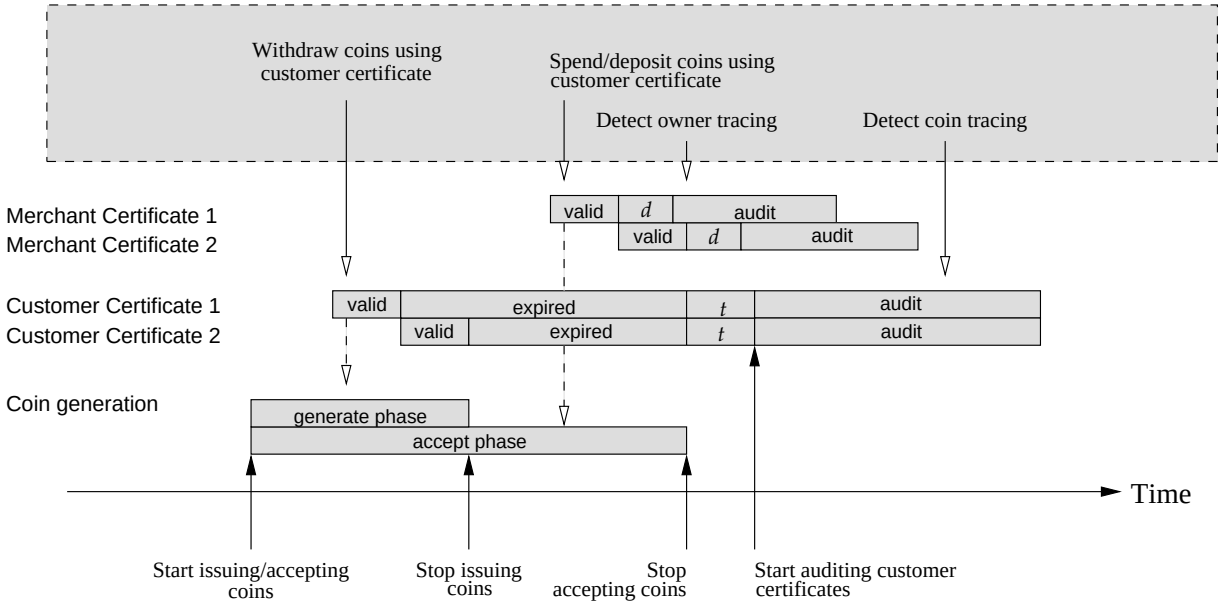
When the bank creates a tracing key (of which the customer is given the public counterpart), it creates a *user certificate*, which contains the user's identity, public tracing key, activation and expiration dates, transaction value limit and a symmetric encryption of the parameter used to generate the tracing key. We hence have *customer* and *merchant certificates*, used to withdraw and deposit coins respectively. Once certificates become invalid, the customer or merchant asks the bank for the symmetric key used to encrypt the key generation parameter, and can request a *tracing certificate* from the bank (a user certificate, signed by a judge, authorising tracing).

The bank issues coins during the *generate phase*, and accepts them for deposit in the *accept phase* (we refer to Figure 2.16 for an example). The activation and expiration dates of any customer certificate must be during the generate phase of a single coin's generation. The audit date for the customer certificate, is always a certain time period d (in Figure 2.16) after the end of the accept phase for this coin's generation. Likewise, the activation and expiration dates of a merchant certificate must be within the accept phase of the coin's generation. The audit date is always a certain time period t after the merchant certificate expires.

In the diagram, coin tracing is undetectable for the time d, and owner tracing undetectable for time t. Authorities can use these time frames to conduct criminal investigations.

Note that the protocol only permits a coin to be traced if the bank *explicitly* enabled coin tracing before its withdrawal. This is something of a disadvantage: not only is *revocable anonymity* not

Figure 2.16 Kügler and Vogt's Auditable Anonymity Revocation Scheme.



possible without *a priori* knowledge of who is to be traced, but auditing of this deanonymisation isn't possible either.

We now briefly describe the setup part of the scheme (Kügler and Vogt, 2003, p. 273–6), which implements a standard ElGamal encryption scheme using a cyclic group G_q, with three pseudo-random generators g_0, g_1, g_2. Given a secret key $x \in \mathbb{Z}_q$ and public keys $y = g^x$, $y_1 = g_1^x$, $y_2 = g_2^x$, the values $G, g, g_1, g_2, y, y_1, y_2$ are published. We now focus only on how coin tracing and minting are implemented. New customers register a public key g_C with the bank, where $g_C = g^{x_s} : x_s \in_R \mathbb{Z}_q$. The bank assigns the customer a public coin tracing key y_C along with a customer certificate, where $y_C = g_2^{x_C} : x_C \in_R \mathbb{Z}_q^*$. Note, however, that if the customer is *not* to be traced, y_C is calculated as $g_C^{x_C}$ instead, meaning that the bank cannot know the discrete logarithm of y_C to the base g_2.

The secret value x_C is encrypted symmetrically in the customer certificate (the authors note that one x_C is used in every generation of coins, for all customers). When x_C is later publicly revealed during the audit period, all customers can determine if their coins are traceable (a customer knows they have been traced if $y_C \neq g_C^{x_C}$). Any customer who have been traced can request a signed certificate authorising this action from the judge The authors note that the tracing key,

used to encrypt the customer's identity, can be proved to have been used correctly using a standard proof of equality of discrete logarithms, due to Chaum and Pedersen (1992).

As the focus of this thesis is not on digital cash, we do not dwell further on how the spending and deposit mechanisms of the protocol work, instead referring the interested reader to the original paper (Kügler and Vogt, 2003). For us, the interesting part of this work is in how the spender's anonymity is revoked: i.e., *if* the customer can show that a key to decrypt her identity was ever created, then she can assume that the bank had this intention. The protocol seems to rely on the honesty of the bank, and on the bank's knowledge of who to trace before coins are even minted (with the exception of double-spenders, who are identified automatically through revelation of a number of values). In electronic voting, this scenario seems inappropriate: double-voting is not the only reason that a voter might need to be traced, and the electoral authorities cannot expect to know who to trace beforehand. We discuss our thoughts on this problem (sometimes identified as the 'digital envelope' problem) in Chapter 6.

2.4 Trusted Computing and the TPM

As we will discuss further in Section 3.2.1, trusted computing is still something of a controversial concept in some circles. A *trusted computer* can be defined as one which, by a number of methods, can prove that it is running certain software, and behaving in a certain manner, without reliance on the end-user.

As noted by Challener et al. (2008), there are many reasons why an authority may wish to be convinced of the trustworthiness of a remote machine: viruses, phishing attacks, badly coded software which is subject to leaks, and eavesdropping are all commonplace. Some of these issues can be mitigated with strong encryption practices, but then, how are the keys used for encryption to be stored? This is one of the founding ideas of trusted computing—it assumes that, at some point, software will be compromised. As such, it provides some trusted *hardware*—the Trusted Platform Module, or TPM—which protects security.

The TPM (the means of providing trusted computing on which we will focus) was designed

by the Trusted Computing Group, and is currently on version 1.2 of its specification (TCG, 2011a,b,c). It ensures the security of private keys and detection of malicious code via three main functions:

- **Public key authentication**. The TPM has a secure random number generator, and functionality to generate keypairs, encrypt, decrypt, sign and verify values. Private keys are generated in the chip, and not accessible outside of it (keys may be stored in external storage, but are encrypted by the TPM).
- **Integrity measurement**. Although private keys are secured, one has to ensure that malicious code can still not access them. Thus, the TPM has a number of *platform configuration registers* (PCRs), which store hashes of configuration values measured at boot. The TPM can be used to encrypt certain values such that they can only be decrypted by a computer whose TPM's PCRs are in a certain state (this is known as sealing).
- **Attestation**. A machine with a TPM is able to use its private keys to sign its PCR values, allowing that machine to prove its trustworthiness to a remote machine. This can be done anonymously, via a protocol known as DAA (Brickell et al., 2004).

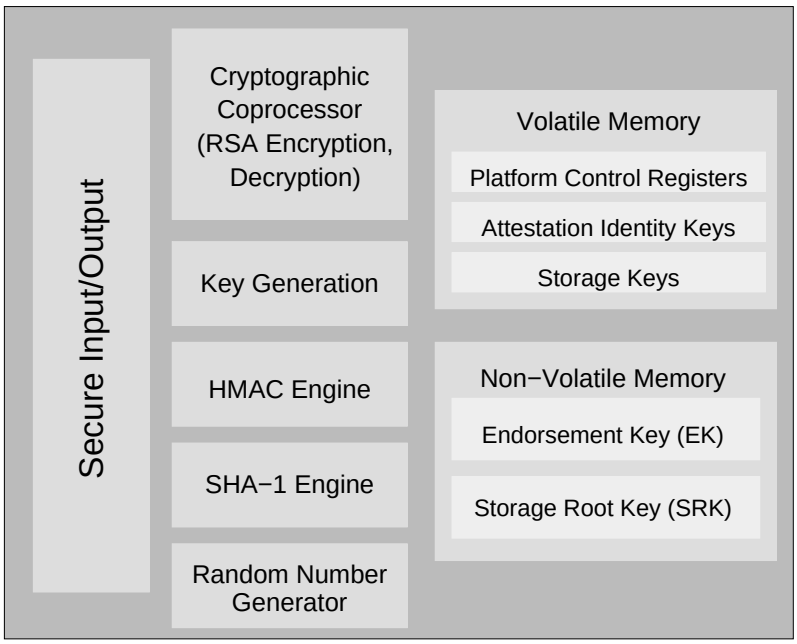
(Challener et al., 2008, p. 10)

2.4.1 TPM Structure

The components of a generic TPM are given in Figure 2.17. Note the two types of storage in the TPM: persistent and volatile. In persistent storage, the *endorsement key* and *storage root key* are stored. The former of these is an RSA keypair created and stored when the TPM is manufactured. The private part of this key is never released from the TPM. The key is used during the assignment of an *owner* to a TPM, when a storage root key is generated: again non-migratable to other TPMs (with some exceptions), this key is the root of a number of keys associated with the protected storage function of the TPM.

A number of other keys can be created by the TPM. Of these, *binding* keys are used to encrypt data such as symmetric keys; *signature* keys are used to create standard RSA signatures,

Figure 2.17 TPM Structure (TCG, 2011a)



and *identity* keys are used in a non-migratable manner, generated inside the TPM, either to sign PCR values or to sign other keys as being non-migratable. For us, Attestation Identity Keys (AIKs) are particularly important, and form a vital part of anonymously identifying a user's TPM in the Direct Anonymous Attestation protocol (see Section 3.2.2 for further information).

A key component of the TPM for our work is the *platform configuration registers* (PCRs). These are in volatile memory, and are used to attest to the state of a machine. When the machine is booted, initially the registers store zeroes, but can then be manipulated via the `TPM_Extend` operation. To extend a PCR by a value x is to concatenate x to the current value of the PCR, and then store the SHA-1 hash of this new value back in the PCR. Because of the hashing algorithm used, it is assumed that one cannot manually force a PCR to be in a certain state, or revert to another state, thus making the registers a sound base from which to verify the state of a machine (Challener et al., 2008, p. 37).

One of the TPM's key functionalities is to encrypt a value such that it can only be decrypted by a TPM whose PCRs are in a defined state (effected by the `TPM_Seal` operation):

> It is possible to infer that a machine is in a state consistent with a certain set of PCR values if it is able to decyrpt a particular value that was locked to that PCR state (Challener et al., 2008, p. 38)

Another of the TPM's operations, `TPM_Quote`, allows the TPM to report the state of a requested PCR, allowing inference of the state of a machine. We use this functionality in our work.

2.4.2 Interaction with the TPM

As is perhaps clear from the previous section, the Trusted Computing Group define a fixed API for interaction with the TPM (TCG, 2011c). The API document is very informative on these commands, and we hence refer the interested user to it. For our purposes, we give an example of a full TPM command, the aforementioned `TPM_Quote`, which returns a signed report of PCR values:

$$\texttt{TPM\_Quote}(tag, paramSize, keyHandle, externalData, targetPCR, authHand,$$
$$authLastNonceEven, nonceOdd, continueAuthSession, privAuth)$$

From the high level at which we interact with the TPM in our work, we are not concerned with the intricate details of each command. As such, we abstract away much of the detail, leaving only the important parts of the command. In the case of the `TPM_Quote` command, we leave only the *externalData* parameter defined, assuming default values for the remaining parameters:

$$\texttt{TPM\_Quote}(\ldots, c_a, \ldots)$$

In our work, we make no alterations or additions to the commands defined in the TPM API.

2.4.3 Summary of TPM Commands Used

In our work in Chapters 5 and 6, we use a subset of the TPM's API commands. Though, for brevity, we do not go into depth about the intricacies of each, in Table 2.8 we explain the purpose of the commands we use. As already discussed, the API commands are the only way in which the operating system and its software can interact with the TPM, which essentially acts as a "black box".

Table 2.8 Summary of TPM Commands Used (TCG, 2011c)

Command	Description
`TPM_DAA_Join`	Begins the *DAA Join* protocol, discussed in Section 3.2.2.3: establishes parameters in the TPM for a particular issuer, allowing the TPM to be certified as a member of a particular group
`TPM_DAA_Sign`	Begins the *DAA Sign* protocol, discussed in Section 3.2.2.3: produces a DAA signature on a message, convincing the message verifier anonymously that the signer was a member of a particular group
`TPM_Quote`	Cryptographically reports requested values from the TPM's PCRs, using a key to sign a statement giving the value of the PCR together with a challenge nonce
`TPM_Extend`	Adds a new measurement to a particular PCR. This is done by hashing the current value of the PCR concatenated with the new value. As a result of the hash function used, it is computationally infeasible to obtain a given PCR value in more than one way
`TPM_CreateWrapKey`	Creates an asymmetric keypair, bound to a specific PCR state (such that a message encrypted—or *bound*—using the public key, can only be decrypted by TPM whose PCRs are in the designated state
`TPM_LoadKey2`	Loads a key into memory for further use
`TPM_Seal`	Encrypts a message using a given key, according to a specific PCR state in which any decrypting platform must be
`TPM_UnSeal`	The reverse of the `TPM_Seal` command, which decrypts a ciphertext only when the TPM's PCRs are in the designated state
`TPM_IncrementCounter`	Increment one of the TPM's built in monotonic counter values

2.5 Summary

In this chapter, we have introduced the requirements for remote electronic voting systems, adding our own requirement for UK-specific elections: namely, revocable anonymity. We have discussed both a variety of electronic voting protocols, and a number of approaches to provision of revocable anonymity in other fields, such as digital cash. Finally, we discussed trusted computing and the TPM, something which we use extensively in Chapters 5 and 6. As noted earlier, no existing work explicitly considers revocable anonymity in electronic voting: this is likely because it is a notion which is not appropriate to many countries' electoral systems. Nevertheless, the UK's legal requirement to be able to link a ballot to a voter motivates us to provide a solution.

One might wonder whether revocable anonymity could simply be "bundled" into an existing electronic voting protocol. That is not the focus of this thesis. Though we could try to alter

existing protocols, we feel that there is no suitable protocol from which to start: for us, the closest protocol to meeting the needs of UK elections is JCJ (Juels et al., 2005), and the system based upon it, Civitas (Clarkson et al., 2008). However, despite considerable research, no secure alternative to the multiple rounds of inefficient plaintext equivalence tests has yet been found, making the protocol inappropriate for wide-scale deployment. JCJ, like many other remote voting protocols, also relies upon the trustworthiness of the machine the voter uses. This is an issue which we address in Chapter 5.

In the next chapter, we discuss the primitives we will use in our work, including cryptographic protocols and primitives, and trusted computing.

3 Preliminaries

Chapter Overview

In this chapter, we introduce a number of primitives and technologies which are needed for the rest of the thesis.

3.1 Cryptographic Primitives

We assume the availability of the following cryptographic primitives. Note that we are working in the formal model, not in provable security. Therefore we make the assumption that the cryptography in the primitives below is perfect.

3.1.1 Threshold ElGamal Encryption Scheme

For the majority of our work, we use a standard ElGamal encryption scheme, under a q-order multiplicative subgroup $G_q = \langle g \rangle$ of $\mathbb{Z}_p^*$, generated by an element $g \in \mathbb{Z}_p^*$, where p and q are suitably large primes, and q divides $(p-1)$. The security of the encryption scheme depends on

the holding of the *Computational Diffie-Hellman* (CDH) assumption in G_q: that is to say that for a chosen cyclic group G of order q, randomly chosen generator g, and $a, b \in_R \{0, \ldots, q-1\}$, given

$$(g, g^a, g^b),$$

it must be computationally intractable to compute g^{ab}. We may infer that ElGamal is *semantically secure* if the *Decisional* Diffie-Hellman (DDH) assumption holds in G: that is to say that for a chosen cyclic group G of order q, randomly chosen generator g and random $a, b \in \mathbb{Z}_q$, given g^a, g^b, the value g^{ab} must be indistinguishable from any other value $g^c \in \mathbb{Z}_q$.

Both the CDH and DDH assumptions are stronger forms of the *discrete logarithm assumption*, that given a generator g, large prime p, and a value x, derivation of e such that $g^e \equiv x \mod p$ is computationally hard (Schneier, 1996, p. 540).

We now explain how encryption works in the ElGamal cryptosystem, and then discuss the differences between decryption and threshold decryption.

3.1.1.1 Key Generation

All agents a in the protocol have a private key $s_a \in \{0, \ldots, q-1\}$ of which only they have knowledge. Each agent has a corresponding public $h_a = g^{s_a}$ where g is a known generator of the subgroup. The public key consists of this h_a along with G, g, q. Public keys are common knowledge to all users.

3.1.1.2 Encryption

In order to encrypt a message m in the encryption scheme we use, a random value $\alpha \in \{0, \ldots, q-1\}$ is selected. The ciphertext is then constructed as

$$(x, y) = (g^\alpha, h^\alpha \cdot m) = (g^\alpha, g^{\alpha s_a} \cdot m)$$

Note that, where the nature of an encryption is unimportant (e.g., where the random exponent, or cryptosystem, is irrelevant), we also denote by $\{m\}_k$ the encryption of a message m with key

k. Now, given the secret key s_a and ciphertext (x, y), the recipient can calculate $g^{\alpha s_a} = x^{s_a}$, and thence m by dividing y by this value. Without the secret key, the calculation of a discrete logarithm is required in order to decrypt y.

3.1.1.3 Threshold Decryption

In our work, we use a (t, n)-threshold decryption scheme analogous to that of Cramer et al. (1997), which uses earlier work by Pedersen (1991). The objective of a threshold cryptosystem is clear:

> …to share a private key among a set of receivers such that messages can only be decrypted when a substantial set of receivers cooperate (Cramer et al., 1997, p. 5)

The problems solved, then, are the joint generation of a secret key by multiple, mutually distrusting parties, and thence the joint decryption of a ciphertext. Joint key generation (Pedersen (1991) in Cramer et al. (1997)) involves each authority A_j being dealt a share $s_j \in \mathbb{Z}_q$ of a secret value s. Public commitments $h_j = g^{s_j}$ are released. The secret s can be recovered from any set Λ of $t < n$ shares (a quorum of t out of n members), using Lagrange coefficients $\lambda_{j,\Lambda}$:

$$s = \sum_{j \in \Lambda} s_j \lambda_{j,\Lambda}; \quad \lambda_{j,\Lambda} = \prod_{l \in \Lambda \setminus \{j\}} \frac{l}{l - j}$$

The public key $h = g^s$ is publicly announced, and no single participant therefore learns s.

To decrypt a ciphertext (x, y), each participant broadcasts $w_j = x^{s_j}$ and a zero knowledge proof that $\log_g h_j = \log_x w_j$. A quorum of legitimate authorities is able to work together to recover the message m as

$$m = y / \prod_{j \in \Lambda} w_j^{\lambda_{j,\Lambda}}$$

In our work, we use a threshold cryptosystem when decrypting votes, to ensure that only a suitably-sized quorum of authorities can decrypt a ciphertext. Combined with our choice of cryptosystem, this means that no single vote is observed by any tallier. An interesting question is how many of the set of authorities constitutes a quorum (i.e., how many must cooperate in order to decrypt a ciphertext). As Benaloh (2006) notes, the number should be high enough to prevent

a small subset colluding to break privacy, but low enough that a small number of "discontented trustees" are still able to prevent the completion of an election by not cooperating. Clearly, any quorum size needs to be determined empirically before the election.

3.1.1.4 ElGamal as a Homomorphic Cryptosystem

It is an important factor of our work that the ElGamal cryptosystem is a *multiplicative homomorphic cryptosystem*. This is to say that given two ciphertexts:

$$(x_0, y_0) = (g^{\alpha}, g^{s\alpha} m_0) \quad (x_1, y_1) = (g^{\beta}, g^{s\beta} m_1)$$

the product of those ciphertexts, *viz.* $(X, Y) = \prod_{i=0}^{1} (x_i, y_i)$, is equal to

$$\begin{aligned}(X, Y) &= (g^{\alpha} \cdot g^{\beta}, g^{s\alpha} m_0 \cdot g^{s\beta} m_1) \\ &= (g^{\alpha+\beta}, g^{s(\alpha+\beta)}(m_1 \cdot m_2))\end{aligned}$$

i.e., the product of any number of ciphertexts is equal to the encryption of the product of the plaintexts. As with much work in electronic voting, we use this to our advantage: choosing the form of the plaintext carefully allows secure tallying of votes, such that it is possible to accurately determine the tally, whilst not showing how any one voter voted. We discuss this more in Chapters 4 and 5.

3.1.2 Strong Designated Verifier Signature Scheme

Designated Verifier signatures and proofs have a long history (Jakobsson et al., 1996), and extensive use in the field of e-voting. We adopt the designated verifier signature scheme of Saeednia et al. (2003) due to its efficient nature, but others would be acceptable. We use designated verifier signatures to enable a prover (Bob, or any one of the first-round tellers in our case) to prove a statement to a verifier (Alice) by proving the validity of a signature:

> Let $P(B, A)$ be a protocol for [Bob] to prove the truth of the statement Ω to [Alice].

> We say that $P(A, B)$ is a strong designated verifier proof if anybody can produce identically distributed transcripts that are indistinguishable from those of $P(A, B)$, except for Bob (Saeednia et al., 2003, p. 43)

However, Alice is unable to prove the signature's validity to *anyone* else, on the grounds that she could have produced it herself (Saeednia et al., 2003, p. 43).

The parameters for the scheme are the same as those for ElGamal encryption: large primes p and q, such that $q|(p-1)$; a generator $g \in \mathbb{Z}_p^*$ of order q, and a one-way hash function hash that outputs values in $\mathbb{Z}_q$. Every user a has a secret key s_a and a corresponding public key $h_a = g^{s_a} \mod p$.

In order to generate a designated verifier signature on a message m for Alice, Bob selects $k \in_R \mathbb{Z}_q$, $e \in_R \mathbb{Z}_q^*$ and calculates

$$c = h_{\text{Alice}}^k$$
$$r = \text{hash}(m, c)$$
$$v = ke^{-1} - rs_{\text{Bob}} \mod q$$

The triple (r, v, e) is now the signature of m (Saeednia et al., 2003). We denote this designated verifier signature as $\text{DVSign}_{\text{Bob}\to\text{Alice}}(m)$, where the signature is generated by i.

Alice is able to verify the signature's correctness by checking the equation

$$\text{hash}(m, (g^v h_{\text{Bob}}^r)^{es_{\text{Alice}}} \mod p) \stackrel{?}{=} r$$

Clearly, no-one other than Alice can verify the signature (verification uses her private key). However, Alice is able to select a random $v' \in \mathbb{Z}_q$, $r' \in \mathbb{Z}_q^*$ and simulate the entire transcript herself (the reader is directed to the authors' paper for proof of this). As a result, Alice could have generated the signature herself, and no third party will be convinced of the validity of any signature from Bob that Alice claims to be valid (Saeednia et al., 2003, p. 45).

Even if Alice reveals her secret key to a third party, she cannot convince that party of the validity of the signature, since she herself could have simulated the signature transcription. Further, no party is able to reveal the contents of the signature without Alice's secret key.

3.1.3 Proof of Equality of Discrete Logarithms

In order to prevent an attack in our voting scheme, we require that the voter demonstrates to a verifier that her vote is of the correct form (without revealing what the vote is).

As we discuss later, a voter's vote is of the form $(x, y) = (g^{\alpha}, h_{\mathbb{T}_2}^{\alpha} g^{M^{i-1}})$ where $\alpha \in_R \mathbb{Z}_q$, M is the maximum number of voters and i represents the position in the list of candidates of the voter's chosen candidate. Alice needs to prove, in zero knowledge, that she is sending to the bulletin board some value for y where the exponent of g is in $\{M^0, \ldots, M^{L-1}\}$ where L is the number of candidates. If we did not have such a proof, any voter could spoil the election by adding spurious coefficients to the exponent, thereby voting several times.

We hence show that the ballot (x, y) is of valid form, as specified in the parameters of the election:

$$(x, y) = (g^{\alpha}, h_{\mathbb{T}_2}^{\alpha} m) : \ m \in \{g^{M^0}, \ldots, g^{M^{L-1}}\}$$

Cramer et al. (1997) demonstrate this via a witness indistinguishable proof of knowledge of the relation:

$$\log_g x = \log_{h_{\mathbb{T}_2}} (y/m_0) \vee \log_g x = \log_{h_{\mathbb{T}_2}} (y/m_1)$$

for an election with only two candidates (Cramer et al., 1997).

This *proof of validity* is described for an interactive, two-candidate scenario in Cramer et al. (1996), Cramer et al. (1997) and Hevia and Kiwi (2004). Using the Fiat-Shamir Heuristic (Fiat and Shamir, 1986), the authors convert the interactive protocol into a non-interactive one (Cramer et al., 1997). However, the scheme provided in these papers is for votes with only two possible outcomes.

The proof of validity for a two-candidate scenario, where a vote $(x, y) = (g^{\alpha}, h^{\alpha} m) : m \in \{m_0, m_1\}$ and the prover knows the value of m, holds, proving that (x, y) is of the proscribed form,

providing Alice submits a vote $v \in \{m_0, m_1\}$, as it provides a witness-indistinguishable proof for the relation given above. The prover knows a witness for either the left or right part, according to the choice of m.

We can extend the two-candidate scenario to L candidates, providing a proof for the relation given by

$$\log_g x = \log_{h_{\mathbb{T}_2}}(y/g^{M^0}) \vee \ldots$$
$$\ldots \vee \log_g x = \log_{h_{\mathbb{T}_2}}(y/g^{M^{L-1}})$$

We adapt the non-interactive proof of ballot validity to a scheme for a *multi-candidate* election. In Figure 3.1, we give a generalised adaptation (**G-PEQDL**) of the above proof of equality of discrete logarithms scheme where Alice votes for candidate k $(1 \leq k \leq L)$ with $(x, y) = (g^\alpha, h^\alpha g^{M^{k-1}})$. This is the only place where we extend one of the primitives we use, and as such we provide a proof, analogous to that of Cramer et al. (1997), for our extension. We note that similar proofs for 1-out-of-L election schemes already exist (Lee and Kim, 2000; Hirt and Sako, 2000)—our work was merely developed independently to suit our own voting protocols. We use the notion of a *binding* encryption scheme (Cramer et al., 1996). A *binding* encryption scheme is one in which any encryption can be opened only *one way*.

Theorem 1. (Security of Generalised PEQDLs) Under the discrete logarithm assumption, the encryption scheme we use is binding in that an encryption can be opened in only one way. Furthermore, the G-PEQDL is a witness-indistinguishable proof of the relation given by

$$\log_g x = \log_{h_{\mathbb{T}_2}}(y/g^{M^0}) \vee \ldots \vee \log_g x = \log_{h_{\mathbb{T}_2}}(y/g^{M^{L-1}})$$

Proof. If an encryption of a ballot can be opened in two different ways, i.e., if for an encryption pair $(x, y) = (g^\alpha, h^\alpha m)$, values α, α' and v, v' can be presented such that $y = h^\alpha g^v = h^{\alpha'} g^{v'}$, where $\alpha \neq \alpha'$ and $v \neq v'$, it follows that $\log_g h = \dfrac{v - v'}{\alpha - \alpha'}$, which contradicts the Discrete Logarithm Assumption.

Like the proof provided by Cramer et al. (1997) for their two-candidate vote scenario, our generalised proof of equality—essentially the same as the two-candidate scenario expanded to L candidates—is, by the results of Cramer et al. (1994), a witness-indistinguishable proof of knowledge that the voter knows $\log_g x$, described by the relation above. The G-PEQDL is hence

specially sound: i.e., given two conversations between prover and verifier in which the initial a_i, b_i messages remain the same, but the challenge, and thus d_i, r_i messages change, a witness can be extracted for each conversation in polynomial time. In particular, the d_k and r_k can always be chosen such that verification succeeds for any value of the hash function challenge c.

Further, the protocol is *special honest verifier zero-knowledge*: i.e., given an honest verifier, and any randomly chosen challenge c, the protocol produces a conversation indistinguishable from the space of all conversations between honest prover and honest verifier in which c is the challenge. We first consider the interactive version of the proof, in which c is chosen uniformly at random by the verifier, and sent to Alice after she sends $(x, y, a_1, \ldots, a_L, b_1, \ldots, b_L)$. Alice replies with $(d_1, \ldots, d_L, r_1, \ldots, r_L)$, and the verification proceeds as in the non-interactive version. Honest verifier zero-knowledge holds because, for random c and $d_i : 1 \leq i \leq L; i \neq k$, and random $r_i : 1 \leq i \leq L$, we can simulate a conversation between the honest verifier and the prover, by choosing a_i and b_i to satisfy the equations given in Figure 3.1. Since c can be chosen freely in the interactive version, we get special honest verifier zero knowledge. The Fiat-Shamir heuristic used to produce the non-interactive version preserves this property. □

We now demonstrate an execution trace of the **G-PEQDL** protocol. Referring to the protocol (Figure 3.1), Alice generates the values α, ω and r_i, d_i (for $i = 1, \ldots, k-1, k+1, \ldots, L$) at random, where she has voted for the k^{th} candidate out of L.

She uses $x = g^{\alpha}, y = h_{\mathbb{T}_2}^{\alpha} g^{M^{k-1}}$ as with her actual vote (note that the value of α does not change), and proceeds to generate $a_k = g^{\omega}, b_k = h_{\mathbb{T}_2}^{\omega}$. All other a_i for $1 \leq i \leq L$ are calculated as $g^{r_i} x^{d_i}$, and all other $b_i \leftarrow h_{\mathbb{T}_2}^{r_i} (\frac{y}{g^{M^i - 1}})^{d_i}$.

Finally Alice generates

$$c \leftarrow \mathsf{hash}(h_{\mathsf{Alice}}, x, y, a_1, b_1, \ldots, a_L, b_L)$$

Using this value she can generate d_k by subtracting all other d_i values from c, and finally $r_k \leftarrow \omega - \alpha d_k$. Alice sends all a, b, d and r values to the verifier.

The verifier now generates c in the same way (note that this value is not sent to him), and trivially this c should equal the sum of all d_i, as Alice manipulated d_k to make this the case. The verifier now checks each value of a_i, b_i:

1. For all $a_{i\neq k}, b_{i\neq k}$: a_i trivially equals $g^{r_i}x^{d_k}$ and b_i trivially equals $h_{\mathbb{T}_2}^{r_i}(\frac{\gamma}{g^{M^{i-1}}})^{d_i}$, as these values were calculated in the same manner by Alice

2. For a_k:

   (a) The value Alice calculates is $a_k = g^{\omega}$

   (b) The verifier makes the comparison:

$$\begin{aligned} a_k &\stackrel{?}{=} g^{r_k}x^{d_k} \\ &\stackrel{?}{=} g^{\omega-\alpha d_k}g^{\alpha d_k} \\ &\stackrel{?}{=} g^{\omega-\alpha d_k+\alpha d_k} \\ &\stackrel{?}{=} g^{\omega} \end{aligned}$$

   which succeeds if Alice is honest.

3. For b_k:

   (a) The value Alice calculates is $b_k = h_{\mathbb{T}_2}^{\omega}$

   (b) The verifier makes the comparison:

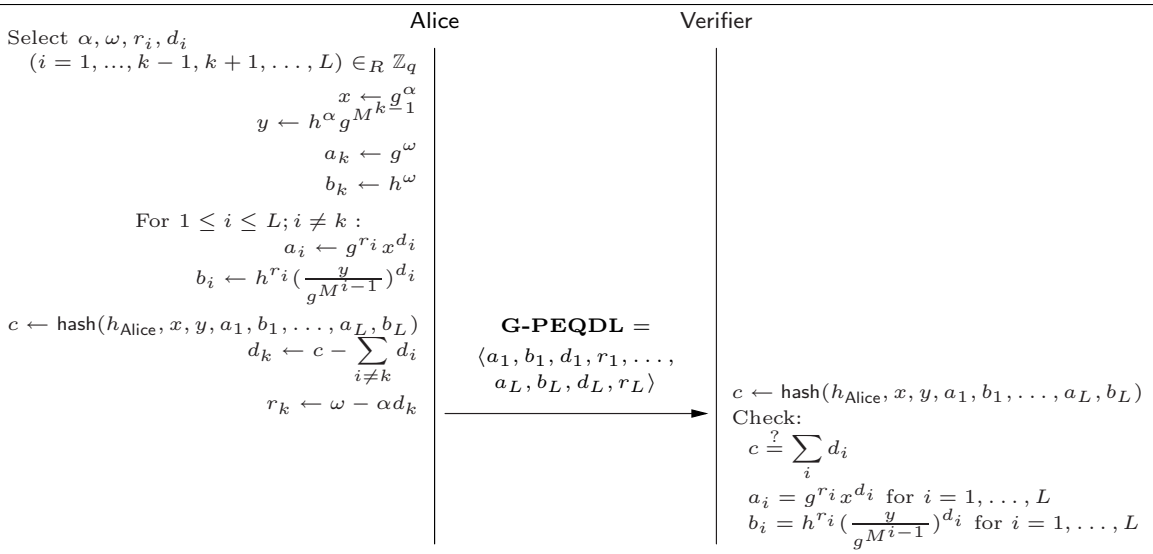
$$\begin{aligned} b_k &\stackrel{?}{=} h_{\mathbb{T}_2}^{r_k}(\frac{\gamma}{g^{M^{k-1}}})^{d_k} \\ &\stackrel{?}{=} h_{\mathbb{T}_2}^{\omega-\alpha d_k}(\frac{h_{\text{Alice}}^{\alpha}g^{M^{k-1}}}{g^{M^{k-1}}})^{d_k} \\ &\stackrel{?}{=} h_{\mathbb{T}_2}^{\omega-\alpha d_k}h_{\mathbb{T}_2}^{\alpha d_k} \\ &\stackrel{?}{=} h_{\mathbb{T}_2}^{\omega-\alpha d_k+\alpha d_k} \\ &\stackrel{?}{=} h_{\mathbb{T}_2}^{\omega} \end{aligned}$$

   which succeeds if Alice is honest.

The only way that Alice could be dishonest to her advantage is to make γ equal to (for example) $h^{\alpha}g^{30M^{k-1}}$, thereby voting 30 times. Similarly she could attempt to vote for more than

one candidate ($y = h^{\alpha} g^{M^{k-1}+M^{k-2}}$), or could attempt to harm the voting tally of a candidate, whilst voting also for her own ($y = h^{\alpha} g^{10M^{k-1}-9M^{k-2}}$). These attacks cannot work. In the verification phase, b_k could not be equal to $h_{\mathbb{T}_2}^{r_k}(\frac{y}{g^{M^{k-1}}})^{d_k}$ if the value of the exponent of g in y is not in $\{M^0, M^1, \ldots, M^{L-1}\}$. Thus, one or more calculations for b_i would fail, and the proof would be rejected.

Figure 3.1 Our generalised non-interactive proof of ballot validity for a vote for candidate k



3.1.4 Designated Verifier Re-encryption Proofs

The properties of the ElGamal encryption scheme allow re-encryption (randomisation) of ciphertexts. Given a ciphertext (x, y), another agent is able to generate $(x_f, y_f) = (xg^{\beta}, yh^{\beta}) : \beta \in_R \mathbb{Z}_q^*$. It is known that given two ElGamal ciphertexts, without knowledge of the private key or the re-encryption factor β, determining any re-encryption relationship between the ciphertexts is hard under the DDH assumption.

In our protocol, we use an ElGamal re-encryption to preserve the voter's anonymity. However, the voter needs to have some conviction that her vote has been counted (individual verifiability). We achieve this via a *Designated Verifier Re-encryption Proof* (DVRP): such a proof convinces Alice that a given re-encrypted ciphertext is equivalent to that she generated, whilst not convincing any third party. We adopt the scheme used by Lee et al. (2004); Lee and Kim (2002) and Hirt and Sako (2000): if $(x, y) = (g^{\alpha} h^{\alpha} m)$ is a ciphertext of a message m as described

above, $(x_f, y_f) = (xg^\beta, yh^\beta)$ is a re-encryption of (x, y). The prover, P (the agent that does the re-encryption) needs to demonstrate to Alice that (x_f, y_f) is equivalent to (x, y) in such a manner that m is not revealed, and this proof is not transferable. P therefore does the following:

1. Selects $d, j, u \in_R \mathbb{Z}_q$
2. Calculates $(a, b) = (g^d, h_P^d)$ and $\sigma = g^j h_{\text{Alice}}^u$, where h_{Alice} is Alice's public key as before.
3. Calculates $c = \text{hash}(a, b, \sigma, x_f, y_f)$ and $z = d - \beta(c + j)$
4. Sends (c, j, u, z) to Alice

Alice then merely needs to verify that

$$c = \text{hash}(g^z (\frac{x_f}{x})^{c+j}, h^z (\frac{y_f}{y})^{c+j}, g^j h_{\text{Alice}}^u, x_f, y_f)$$

As detailed by Hirt and Sako (2000), Alice is able to generate this proof for herself as she knows her own private key (σ is a trapdoor commitment for j and u), meaning that no-one (other than Alice) can be convinced by it. Indeed, Alice can insert 'fake' proofs into any communication meant for her, to fool observers.

3.1.5 Signature Scheme

We assume the availability of a standard signature scheme in our work. Though we could use the ElGamal signature scheme described in ElGamal (1985), it is rarely used in practice. Instead we suggest the Digital Signature Algorithm (National Institute of Standards and Technology, 2009), based on the original ElGamal scheme.

Parameters The system requires a hash function hash. We select a value p of length L, where L is a multiple of 64; q, a prime factor of $p - 1$ of length N (between 160 and 256 bits, dependent on the value of L); $g = h^{(p-1)/q} \mod p$, where $h < p - 1$ such that $g > 1$; $x < q$ and $y = g^x \mod p$. The values p, g, q are public; the private key is x and the public key is y.

Signature Generation To sign a message m, Alice selects $k < q$ and generates $r = (g^k \mod p) \mod q$; $s = (k^{-1}(\mathsf{hash}(m) + xr)) \mod q$, then sends the signature, denoted $\mathsf{sign}_{\mathsf{Alice}}(m) = (r, s)$, to Bob.

Verification To verify a signature, Bob computes $w = s^{-1} \mod q$, $u_1 = (\mathsf{hash}(m) \cdot w) \mod q$, $u_2 = (rw) \mod q$, and $v = ((g^{u_1} \cdot y^{u_2}) \mod p) \mod q$. If $v = r$, the signature is valid.

3.1.6 Threshold Signature Scheme

In order to ensure that eligibility and uniqueness are always satisfied in our protocol, we employ a $(t, n)-$threshold signature scheme during the voting phase of the protocol. A threshold signature scheme works in a similar way to a threshold decryption scheme: of n possible talliers, t must collude to generate a signature on a message. The scheme that we adopt is not of great consequence, but the one used by Harn (1994) has good verification properties and fits in well with the exponential ElGamal cryptosystem that we use:

The scheme, like other threshold schemes, is based on the perfect secret sharing scheme of Shamir (1979). We begin with some values agreed by all group members:

1. p, a large prime modulus;
2. q, a prime divisor of $p - 1$;
3. α, where $\alpha = h^{(p-1)/q} \mod p$, h is a random integer between 1 and $p - 1$ and $\alpha > 1$. α is a generator of order q in GF(p). The values p, q, α are the public values; the a_i values are secret.

3.1.6.1 Key Generation

Each member i selects two integers, z_i, x_i at random from $[1, p-1]$ and computes a public key as $y_i = \alpha^{z_i} \mod p$. The pair x_i, y_i are then that member's public key, with z_i his secret key. Each member then needs to use the same pre-agreed $(t, n-1)-$secret sharing scheme to distribute his key to the other group members. To do this, member i selects a $(t-1)^{\text{th}}$-degree polynomial $f_i(x)$,

where $f_i(0) = z_i \mod q$, and computes $f_i(x_j) \mod q$, and the corresponding public $y_{i,j} = \alpha^{f_i(x_j)} \mod p$, for each other member u_j.

3.1.6.2 Signature Generation

If considering a $(t, n)-$threshold encryption system, we note that a threshold (t)-sized quorum of members is required to collude in order to decrypt a ciphertext. In a threshold signature scheme, t members, indexed $1, 2, \ldots t$, are required in order to sign a message. Each member i begins by signing the message using a secret key which they choose at random, $k_i \in_R [1, q-1]$, and computes a public value r_i as $r_i = \alpha^{k_i} \mod p$. The public value r_i is distributed publicly.

When all of these r_i values are available, each member can calculate

$$r = \prod_{i=1}^{t} r_i \mod p$$

Each member i can then use his own z_i and k_i, and the secret $f_j(x_i)$ for $j = t+1, t+2, \ldots, n$ to sign the message m, by solving the equation

$$s_i = \left\{ z_i + \sum_{j=t+1}^{n} f_j(x_i) \cdot \left(\prod_{k=1, k\neq i}^{t} \frac{-x_k}{x_i - x_k} \right) \right\} \cdot m' - k_i \cdot r \mod q$$

for $s_i \in \mathbb{Z}_q$, $m' = f(m)$. An assembly clerk (or, indeed, the signature's intended recipient) is sent $\{m, s_i\}$, where $\{r_i, s_i\}$ is i's partial signature on m. When the clerk has received an $\{r_i, s_i\}$, he validates its authenticity using i's public signature keys, as well as $y_{j,i}$ for $j = t+1, t+2, \ldots, n$ to assert the equality

$$\left\{ y_i \left(\prod_{j=t+1}^{n} y_{j,i} \right) \prod_{k=1, k\neq i}^{t} \frac{-x_k}{x_i - x_k} \right\}^{m'} = r_i^r \alpha^{s_i} \mod p$$

where $m' = f(m)$.

When t partial signatures are received by the clerk, the full signature can be generated as $\{r, s\}$, where r was already calculable and $s = \sum_{i=1}^{t} s_i \mod q$.

3.1.6.3 Signature Verification

As with a non-threshold scheme, the signature can be verified by anyone with access to the group public key y, by asserting the equality

$$y^{m'} = r^r \alpha^s \mod p$$

3.2 Other Preliminaries

3.2.1 Trusted Computing

The concept of *trusted computing* is still rather controversial in some circles. A trusted computer is one that, through the use of a *trusted platform module* (TPM), and other technologies such as memory curtaining, sealed storage and *remote attestation*, removes reliance on the end user to prove that his computer contains a tamper-resistant module, with which communications can be trusted to be authentic. The TPM is designed to be tamper-proof. The benefits of its use for remote applications requiring secure information flow and data handling are clear.

In the field of remote electronic voting (that is, voting from any internet-connected terminal), for example, we might require that a user can only vote from a machine that is running the correct voting software, for obvious reasons. We could do this by providing each voter with a bootable operating system 'live CD'-type disc[1].

However, we naturally still require that the voter using the trusted machine remains *anonymous*, whilst still being able to demonstrate that the machine she is voting from is trustworthy.

Interaction with the TPM is permitted only through a list of predefined commands, given in the TPM's *API*, as discussed in Section 2.4. We do not modify these commands in any way, and denote the use of one of them `as such`. TPM commands are generally invoked directly by the host machine, or by a remote machine via an *encrypted transport session*. For brevity we do

[1] We note that, as suggested by Fink et al. (2009), security of any system that obtains software and private keys from removable media is vulnerable to compromise. This issue can be mitigated by having the TPM compare a publicly known signed hash of the intended executable code with a hash the TPM itself generates. In fact, the user could make this comparison.

not elaborate on the structure of, or commands of the TPM API here. The reader is directed to (TCG, 2011a,b,c; Challener et al., 2008) for further information. For now, it is sufficient to state that actions performed by the TPM are trustworthy.

3.2.2 Direct Anonymous Attestation

Attestation in our context is the idea is that some verifier wishes to be convinced that Alice is using a machine which contains a valid, permitted TPM, and that (later) this TPM can prove that Alice's machine is running the correct software. However, the identity of the user or of the specific TPM should not be revealed, as this would make her transactions linkable (Brickell et al., 2004). As stated by Brickell et al. (2004), a possible solution to this problem was to give all TPMs the same keypair to sign and encrypt, thereby making all transactions indistinguishable. Of course, this solution would never work.

Hence, the Trusted Computing Group's first solution was to use a Privacy Certification Authority as a trusted third party for every authentication. This naturally introduces problems: the CA would have to be permanently available, and it would have to be trusted.

Direct Anonymous Attestation (DAA) is the solution that the TCG accepted, and is currently built into the TPM specification. The DAA protocol is complex, and we advise that the uninitiated reader consult Brickell et al. (2004) for a full explanation. On a high level, DAA is split into three sub-protocols: *join*, *sign* and *verify*. In the *join* protocol, a host and a TPM gain *attestation* on a secret value, chosen by the TPM, demonstrating that the host's machine contains a valid TPM. In the *sign* protocol, the host and TPM anonymously prove that they gained this attestation, and produce a DAA Signature on some message (generally a key). This signature is verified in the final stage of the protocol.

For our purposes, we use DAA as follows. Alice, wishing to vote, engages in the DAA *join* protocol with the registrar, $\mathbb{R}$, using a pseudonym $N_{\mathbb{R}}$ randomly generated by her TPM. $\mathbb{R}$ will check that Alice is eligible to vote, and then issue her a certificate proving this fact.

Once Alice has obtained her certificate in our *join* protocol, she is free to (at any time afterwards) form a different pseudonym $N_{\mathbb{T}}$ with which to vote, and then authenticate herself to $\mathbb{T}$

using the DAA *sign* protocol. $\mathbb{T}$ will employ the DAA *verify* protocol, and then require that Alice proves the state of her machine in some way, and, providing her machine's state is correct, will accept her vote for further processing.

The Direct Anonymous Attestation protocol (Brickell et al., 2004) is used to allow a remote Trusted Platform Module (TPM) to anonymously attest to the state of the machine in which it resides. In our context, it is particularly important that the attestation *is* anonymous—it is with this feature that we allow a voter to vote. The DAA protocol uses a novel technique to detect duplicate requests (in our case, votes), and to blacklist rogue TPMs.

We first point out some of the notation used by the authors (Brickell et al., 2004), and then discuss the DAA protocol in more detail.

3.2.2.1 Notation

In order to permit the selection of the high and low order bits of some integer x, the authors denote $\mathtt{LSB}_u(x)$ to be $x - 2^u\lfloor\frac{x}{2^u}\rfloor$, and $\mathtt{CAR}_u(x)$ to be $\lfloor\frac{x}{2^u}\rfloor$, for some u. For example, if $(x_k \ldots x_0)_b$ is the binary representation of $x = \sum_{i=0} k2^i x_i$ (e.g., $(1100)_b$ is the binary representation of 12), then $\mathtt{LSB}_u(x)$ is the integer corresponding to the u least significant bits of $(x_k \ldots x_0)$. $\mathtt{CAR}_u(x)$ is the binary representation of x, right-shifted by u bits. Note hence that $x \in \mathbb{N} = \mathtt{LSB}_u(x) + 2^u\mathtt{CAR}_u(x)$. It is these functions that are later used to split a value f into two parts, f_0 and f_1.

A common scheme is adopted to represent proofs of knowledge, without discussing the detail of those proofs. The line

$$PK\{(\alpha, \beta, \gamma) : y = g^\alpha h^\beta \wedge \tilde{y} = \tilde{g}^\alpha \tilde{h}^\gamma \wedge (u \leq \alpha \leq v)\}$$

represents a zero-knowledge proof of knowledge of $\alpha, \beta, \gamma \in \mathbb{N}$, such that $y = g^\alpha h^\beta$ and $\tilde{y} = \tilde{g}^\alpha \tilde{h}^\gamma$, where $u \leq \alpha \leq v$ (Brickell et al., 2004).

3.2.2.2 The CL-Signature Scheme

The DAA scheme is based on the Camenisch-Lysyanskaya (CL) signature scheme (Camenisch and Lysyanskaya, 2003). The scheme is mentioned here for completeness. A special RSA modulus $n = pq, p = 2p' + 1, q = 2q' + 1$ (p, q, p', q' all prime) is chosen, followed by $R_0, \ldots, R_{L-1}, S, Z$ from the group of quadratic residues modulo n, QR_n. The secret key is p; the output public key is $(n, R_0, \ldots, R_{L-1}, S, Z)$.

Given that ℓ_m is a parameter, and the messages space is the set $\{(m_0, \ldots, m_{L-1}) : m_i \in \pm\{0,1\}^{\ell_m}\}$, on input of a message $m = m_0, \ldots, m_{L-1}$, a random prime e is chosen with a random number v, then the value A is computed such that $Z \equiv R_0^{m_0} \ldots R_{L-1}^{m_{L-1}} S^v A^e \mod n$ (this is a generalisation of the way in which A is calculated in DAA, shown in Section 3.2.2.3). The signature on m is then (A, e, v). Verification is via checking that this equivalence holds for a given message.

3.2.2.3 The DAA Protocol

The protocol has three participants: the *Host* (who, working with her TPM, attests to the state of her machine); the *Issuer*, who permits the host to *join* (gain attestation), and the *Verifier*, who determines whether the host indeed gained certification.

Issuer Setup

1. The issuer chooses an RSA modulus, $n = pq$ as described above, and a generator g' of QR_n. It selects $x_0, x_1, x_z, x_s, x_h, x_g \in [1, p'q']$ and generates:

$$g := g'^{x_g},\ h := g'^{x_h},\ S := h^{x_s},$$
$$Z := h^{x_z},\ R_0 := S^{x_0},\ R_1 := S^{x_1}$$

   (all mod n).

2. The issuer proves non-interactively that the values above are calculated correctly (proving that each of the values in the key lie in the correct subgroups), then generates a group of prime order, by choosing primes ρ, Γ such that $\Gamma = r\rho + 1$ for some r with ρ which

does not divide r (more details are given in the paper). Next it chooses $\gamma' \in_R \mathbb{Z}_\Gamma^*$ and sets $\gamma = \gamma'^{(\Gamma-1)/\rho} \mod \Gamma$.

3. The public key is declared as $(n, g', g, h, S, Z, R_0, R_1, \gamma, \Gamma, \rho)$; (p, q) is stored as the private key.

The *Join* Protocol

From a high level, the *join* protocol is where a host (and its TPM) gain attestation on a secret value chosen by the TPM, from an Issuer. Each issuer has a *basename* $\texttt{bsn}_I$ (I for "Issuer", resp. V for "Verifier"), and an associated value $\zeta_I = (H_\Gamma(1 \parallel \texttt{bsn}_I))^{(\Gamma-1)/\rho} \mod \Gamma$, where H, H_Γ each represent hash functions.

The TPM verifies the ζ_I value supplied by the host, by ensuring that $\zeta_I^\rho \equiv 1 \mod \Gamma$, and generates two random values f_0,f_1, and a number of other values, by splitting an initial random secret f:

$$f = H(H(\texttt{DAAseed} \parallel H(PK_I')) \parallel \texttt{cnt} \parallel 0) \parallel \ldots \parallel H(H(\texttt{DAAseed} \parallel H(PK_I')) \parallel \texttt{cnt} \parallel i) \mod \rho,$$

$$f_0 = \texttt{LSB}_{\ell_f}(f);\ f_i = \texttt{CAR}_{\ell_f}(f);\ \nu' \in_R \{0,1\}^{\ell_n + \ell_\varnothing};\ U = R_0^{f_0} R_1^{f_1} S^{\nu'} \mod n;\ N_i = \zeta_I^{f_0 + f_1 2^{\ell_f}} \mod \Gamma$$

`DAASeed` is a random seed used for calculation of f by the TPM, PK_I' is a long-term public key of the Issuer, and `cnt` relates to the number of times the TPM has run the *join* protocol. All values ℓ_x are DAA security parameters, defined further by the authors. i is equal to $\lfloor \frac{\ell_\rho + \ell_\varnothing}{\ell_\mathcal{H}} \rfloor$ (1, with the default security parameter values). N_I is the *pseudonym* the Issuer will know the TPM by. U, N_I are sent to the issuer via the host.

The Issuer checks whether N_I represents a rogue (i.e., non-trustworthy) TPM, and checks whether the pseudonym has been used before. If not, the TPM and issuer engage in a signature proof of knowledge of the values (f_0, f_1, ν'). The authors represent such a proof of knowledge in a high-level manner, where the protocol

$$SPK\{(f_0, f_1, \nu') :\ U \equiv \pm R_0^{f_0} F_1^{f_1} S^{\nu'} \mod n \ \wedge\ N_I \equiv \zeta_I^{f_0 + f_1 2^{\ell_f}} \mod \Gamma$$
$$\wedge\ f_0, f_1 \in \{0,1\}^{\ell_f + \ell_\varnothing + \ell_\mathcal{H} + 2} \ \wedge\ \nu' \in \{0,1\}^{\ell_f + \ell_\varnothing + \ell_\mathcal{H} + 2}\}(n_t || n_i)$$

represents a signature proof of knowledge of the values f_0,f_1, ν' such that the assumptions given are

satisfied, using the nonces n_i and n_t. If successful, the Issuer chooses $\hat{v}$ and a prime e at random, and computes $v'' = \hat{v} + 2^{\ell_v - 1}$ (ℓ_v is a security parameter), and

$$A = \left(\frac{Z}{US^{v''}} \right)^{1/e} \mod n$$

sending (A, e, v'') to the host with a proof of A's correctness, in the form of a proof of knowledge of d such that $d \equiv \pm(\frac{Z}{US^{v''}})^d \mod n$. The host forwards v'' to the TPM, which calculates $v = v'' + v'$ and stores (f_0, f_1, v).

The *Sign* Protocol

Now that the host and TPM have gained an attestation credential on f_0, f_1, they can prove this to a verifier. The aim is for the platform to sign a message m (in our case, an *Attestation Identity Key*—AIK, used to sign further messages).

We begin again by generating $\zeta_V = (H_\Gamma(0 \parallel \mathtt{bsn}_V))^{(\Gamma-1)/\rho} \mod \Gamma$, where $\mathtt{bsn}_V$ is, in our case, the basename of the talliers. The TPM verifies ζ_V, and the host selects w, r at random, computing

$$T_1 = Ah^w \mod n; \quad T_2 = g^w h^e (g')^r \mod n$$

The TPM computes N_V using the same method as for N_I, then sends it to the host. The host and TPM now produce a signature proof of knowledge that T_1, T_2 commit to a certificate that was computed using N_V.

This done, the host generates a signature

$$\sigma = (\zeta_V, (T_1, T_2), N_V, c, n_t, (s_v, s_{f_0}, s_{f_1}, s_e, s_{ee}, s_w, s_{ew}, s_r, s_{er}))$$

where n_t is a nonce and c is a hash containing the message m, and sends this to the Verifier. The details of the calculation of each s value are given in the original paper. Verification is simply a matter of checking the signature's correctness, the correctness of ζ_V, and whether N_V represents a rogue TPM.

In our protocol, we use (as is suggested in the paper) a fresh Attestation Identity Key (generated by the TPM) as the message *m* which is DAA-signed by the host and the TPM. Once the Talliers have an authenticated copy of this key, the TPM can sign its internal registers to prove their state, and therefore to attest to the state of the machine.

3.2.3 Physical and Virtual Monotonic Counters

In the work we present in Chapter 6, one of the most important capabilities of the TPM is the availability of secure *monotonic counters*. Monotonic counters are tamper-resistant counters embedded in the TPM, which, once incremented, cannot be reverted to a previous value: this reduces the likelihood of replay attacks, for many applications (Sarmenta et al., 2006).

Unfortunately, the 1.2 version of the TPM, being a low-cost piece of hardware, has only four monotonic counters, of which only one can be used in any boot cycle. As noted by Sarmenta *et al.*, the intention here was to implement a higher number of *virtual* monotonic counters on a trusted operating system. Trusted operating systems are a requirement we would rather not enforce, however. The work of Sarmenta et al. (2006) demonstrates the creation of an unbounded number of virtual monotonic counters with a non-trusted OS.

A virtual monotonic counter is a mechanism (in untrusted hardware or software) which stores a counter value, and provides two commands to access it: ReadCounter, which returns the current value, and IncrementCounter, which increases the counter's value. The counter's value must be *non-volatile*, increments and reads must be *atomic*, and changes must be irreversible. Note that *virtual* monotonic counters are not stored on the TPM, but instead on untrusted storage, allowing a far higher number of simultaneous counters to be used.

The manner in which Sarmenta et al. (2006) implement their solution means that the counter is not tamper-*resistant*, but merely tamper-evident. This is sufficient for our purposes, as an attempt to tamper with such a counter can be seen by any observer as an illicit attempt to modify its value. The counter produces *verifiable* output in the form of unforgeable *execution certificates*, via a dedicated attestation identity key (AIK) for each counter. The counter uses this key, together with nonces, to produce signed execution certificates to send to users.

In the implementation of virtual monotonic counters suggested by Sarmenta et al. (2006, p. 31), the counter mechanism is stored in full on the host (rather than on the host's TPM), and supports the following functions:

- CreateNewCounter(nonce): returns a CreateCertificate containing the ID number of the counter, and the nonce given as a parameter
- ReadCounter(CounterID,Nonce): returns a ReadCertificate containing the value of the counter, the counter's ID and the given nonce
- IncrementCounter(CounterID,Nonce): increments the counter, and returns an IncrementCertificate containing the new value of the counter, counter ID and nonce
- DestroyCounter(CounterID,Nonce): destroys the counter.

In this work, we assume availability of the virtual monotonic counters defined by Sarmenta *et al.*. To avoid use of commands that are not included in the TPM API, we adopt the first, log-based scheme which they define (Sarmenta et al., 2006, p. 32). As noted earlier, the TPM has a limited number of physical monotonic counters, of which only one at a time can be used. The log-based implementation of virtual monotonic counters uses a physical monotonic counter as a "global clock", where the time t is simply the value of the physical counter at a given time.

The value of a virtual monotonic counter is then the value of the global clock at *the last time the virtual counter's* IncrementCounter *command was executed*. This consequently means that the value of a counter each time it is incremented cannot be predicted deterministically—we can merely say with certainty that the value of the counter will only monotonically increase. As we discuss further in the Chapter 6, this does not present a problem for us.

The IncrementCounter operation is then implemented using the `TPM_IncrementCounter` TPM API command, inside an exclusive, logged transport session, using the counter's ID and the client's nonce (*viz.* hash(CounterID$||n_S$)) to prevent replay. The result of the final operation, a call to `TPM_ReleaseTransportSigned`, is a data structure including the nonce, and a hash of the transport session log, which is used to generate an IncrementCertificate.

The ReadCounter operation is more complex, and involves the host (idp, for us) keeping an array of the latest increment certificates (Sarmenta et al., 2006, p. 33) for each virtual counter, returning the right one when the client requests it. In order to prevent reversal of the counter's value, however, the host must send the current time certificate, the current increment certificate, and all of the previous increment certificates. Verification of the counter's value then involves checking that each previous increment certificate is *not* for the counter whose ID has been requested.

We do not go into further implementation specifics, but instead refer interested readers to Sarmenta et al. (2006, p. 32) for further information.

3.2.4 Anonymous Channel

Due to the nature of the DAA protocol (Brickell et al., 2004), we need to use of some sort of anonymous channel during the voting phase (not doing so would lead to Alice's pseudonyms being linkable, and her vote therefore being traced). Due to the nature of our work in Chapter 5, we need this channel to be bidirectional, so that Alice can receive proofs of her vote having been counted. We note that standard mix networks are not designed to receive replies, but onion routing-based networks are (Dingledine et al., 2004). Like much work in electronic voting, however, we deliberately do not specify how the anonymous channel is created, but note that it is only important that a user's communications through the channel are anonymous: untappable channels are not required.

3.3 Summary

In this chapter, we have introduced a number of the preliminaries which we require for our work, including our own work on a generalisation of the two-candidate proof of discrete logarithm equality to L candidates. We have extensively discussed the TPM's DAA protocol, which we use in Chapter 5. In the next chapter, we discuss our first protocol on electronic voting with revocable anonymity.

4 Revocable Anonymity in Electronic Voting[†]

Chapter Overview

In this chapter, work on a remote electronic voting protocol providing revocable anonymity to voters, which was presented in December 2009 (Smart and Ritter, 2009), is discussed.

The chapter details a remote electronic voting protocol which satisfies several properties considered important in electronic voting. This leads to two main contributions:

- A secure voting protocol allowing a quorum of authorities to link a ballot to its voter (revocable anonymity), whilst achieving coercion-resistance and legitimate voter privacy
- A novel method of allowing the voter to achieve coercion-resistance without anonymous channels or tamper-resistant devices, through designated-verifier signatures

[†]This chapter is an extended version of work presented at the Fifth International Conference on Information Systems Security (Smart and Ritter, 2009).

The protocol discussed in this chapter achieves the above properties, as well as the standard electronic voting properties (completeness, uniqueness, coercion-resistance, fairness, and legitimate-voter privacy).

4.1 Chapter Structure

In Section 4.2, we give a simple schematic and a high-level description for our first protocol. In Section 4.3, we give the participants, trust model and threat model for our work. We present the protocol in Section 4.4, and provide an analysis of the security properties claimed in Section 4.5. Note that a formal analysis of the protocol is given in Chapter 7.

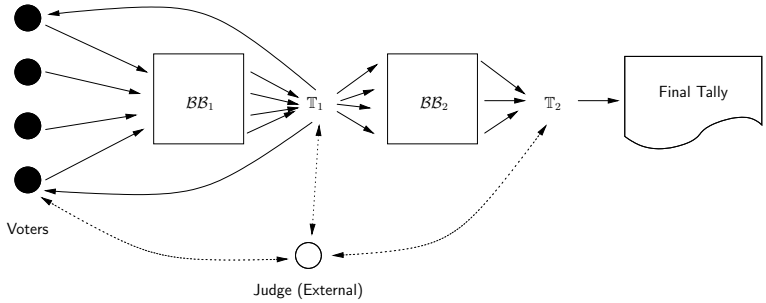
4.2 Protocol Schema

We present a two-phase protocol, where voters do not need to synchronise between phases they are actively involved in. Our reasoning for splitting into two phases is to preserve the anonymity of the legitimate voter, henceforth referred to as Alice. In the first phase, voters receive eligibility tokens with designated verifier signatures, and form ElGamal encryptions of ballots, submitting them to a bulletin board. A member of a semi-trusted tallier group re-encrypts Alice's vote.

In the second phase, Alice receives a designated verifier proof of re-encryption (along with some other fake proofs of re-encryption), and her re-encrypted vote is posted to another bulletin board with an encrypted version of her identity. Alice can then check her vote has been included, or contact a Judge otherwise.

Once all votes are posted to the second bulletin board, a tally is calculated and announced. A simple schematic diagram of the protocol is given in Figure 4.1.

Figure 4.1 A schematic for our protocol.



4.3 Protocol Model

4.3.1 Participants

Our protocol is modelled with 5 kinds of participants. A participant (agent) is an interactive, probabilistic, polynomial-time computation. All agents are able to communicate via a network, which is not secure or anonymous in any way.

The participants are as follows:

- **Voters**. The protocol allows M voters $v_i \in \{v_0, v_1, ..., v_{M-1}\}$ to vote. Alice is an honest voter who wishes to vote anonymously. She is able to vote many times, but *once* unobserved. Eligible voters' public keys are publicly known.

- **First Round Bulletin Board/First Round Talliers**. Our protocol uses two separate *bulletin boards*. A standard bulletin board is a public broadcast channel with memory. The first bulletin board we use is writable only by *voters*. All voters send an encrypted vote and signed proof of validity to this board, which we denote as $\mathcal{BB}_1$.

  The *first-round talliers* $\mathbb{T}_1$ are a *semi-trusted* group of agents[1], each possessing an ElGamal secret key $s_{\mathbb{T}_1}$ in its entirety, which any one of them can use to remove the first layer of encryption on Alice's vote[2]. We assume that each instance would be busy enough, and that votes would be batched before sending to $\mathcal{BB}_2$, so that timing attacks would be

[1] We discuss our need for trusting $\mathbb{T}_1$ later in this Section.

[2] The size of $\mathbb{T}_1$ would need to be determined empirically depending on the size of the electorate. Since each member of the group has a copy of the same key, the size only affects how much of a bottleneck (in terms of computational power) $\mathbb{T}_1$ is.

ineffective. Our justification for having multiple members of $\mathbb{T}_1$ is to prevent a bottleneck of computational power, but if this problem were ignored, we could equally substitute the group for a single entity.

The first round talliers are responsible for ensuring that Alice's vote is valid according to the set of valid possible votes, not coerced, and not a double-vote. They are unable to see Alice's actual vote token. $\mathbb{T}_1$ also encrypts Alice's identity, should anonymity revocation be required. They issue Alice with vote validity tokens during registration.

- **Second Round Bulletin Board/Second Round Talliers**. The second bulletin board $\mathcal{BB}_2$ is viewable by all users of the protocol, and writable only by $\mathbb{T}_1$. It lists only the re-encrypted (valid) votes in a random permutation. The votes themselves, (x, y), are encrypted with the public key of the second round talliers.

  The *second-round talliers* are a group of agents (disjoint from $\mathbb{T}_1$) who decrypt the ballots listed on the second round bulletin board using threshold ElGamal with a shared key $s_{\mathbb{T}_2}$. The second round talliers will also publish the final tally.

- **Anonymity Teller Group**. As well as each being separate groups $\mathbb{T}_1, \mathbb{T}_2$, the tallier groups form part of a larger group which deals only with the voter's anonymity. This group contains an equal number of members of $\mathbb{T}_1$ and $\mathbb{T}_2$ and is simply denoted $\mathbb{T}$. As such, it has a public key $g^{s_{\mathbb{T}}}$ and associated private key $s_{\mathbb{T}}$, where the private key is distributed amongst all members as before. In this case, to decrypt, a quorum of a size t_{id}, greater than the size of either $\mathbb{T}_1$ or $\mathbb{T}_2$, will need to collude to decrypt. Note that this decryption is only ever needed when a voter's identity needs to be traced, as our protocol is optimistic. Further, a voter's anonymity cannot be revoked without the agreement of the quorum *and* the Judge.

- **Judge**. The Judge is an entity of the protocol that is rarely used. She has two purposes:

  1. If Alice cannot find her re-encrypted vote on the public bulletin board, she asks the Judge for verification.

2. The Judge also authorises anonymity revocation (having been presented with appropriate evidence of the need for revocation) in order to deliberately link a ballot to a voter, by applying her private key for a decryption.

Note that the Judge is only used in a minority of cases, i.e., where a voter's identity needs to be revealed, or Alice cannot find her vote on the bulletin board. The Judge, understandably, is trusted. We note that she could equally be formed from a coalition of mutually distrusting parties, disjoint from $\mathbb{T}_1/\mathbb{T}_2$, and selected by the electoral authorities. However, we see the Judge more in terms of a physical arbiter of justice in a court of law.

4.3.2 Trust Model

We make the following assumptions in our protocol:

1. All parties trust that $\mathbb{T}_1$ will not reveal the link between a ballot (x, y) and its re-encryption (x_f, y_f)
2. All parties trust that $\mathbb{T}_1$ will perform valid encryptions of each voter's identity, to afford anonymity revocation
3. The Judge and $\mathbb{T}_2$ trust that $\mathbb{T}_1$ will only sign and post to $\mathcal{BB}_2$ ballots which are valid
4. The Judge trusts that $\mathbb{T}_1$ will accurately and honestly send any data requested by it, to the Judge
5. All participants trust that the Judge will only authorise revocation of anonymity in appropriate circumstances
6. Alice trusts that she will receive one (and only one) valid voting token, along with several invalid ones, from the first-round talliers during registration.
7. Alice trusts the Judge to honestly state whether votes have been counted
8. All parties trust that voter identities will be stored correctly (and securely) on the second-round bulletin board

Note that we have already assumed that: $\mathbb{T}_1$ will batch votes before sending to $\mathcal{BB}_2$, to prevent timing attacks; Alice can vote once unobserved; and a t-sized quorum of $\mathbb{T}_2$ will not collude to break fairness or decrypt ballots until voting is over.

4.3.2.1 (Partially) Trusting $\mathbb{T}_1$

The purpose of the first-round talliers is to check the eligibility of Alice to vote and to re-encrypt Alice's vote before it is posted to the second bulletin board. To achieve anonymity, we need to *partially* trust $\mathbb{T}_1$. This means that we trust that $\mathbb{T}_1$:

- will not reveal the link between Alice's ballot (x, y) and her re-encrypted ballot (x_f, y_f), except by request of the Judge;
- will make valid encryptions of voter identities when forming $\overline{\text{id}}$ tags;
- will act honestly in communications with the Judge (no other honest communications are required than those stated here);
- will only sign and post to $\mathcal{BB}_2$ ballots which are valid

Note therefore that $\mathbb{T}_1$ *at no point* has access to Alice's unencrypted vote. We further do not trust $\mathbb{T}_1$ to reliably send communications—if messages do not arrive as expected, the voter can detect this.

We believe that the trust we have placed in $\mathbb{T}_1$ is the minimum assumption necessary to assure the properties we wish to satisfy.

4.3.3 Threat Model

In this section, we consider the potential threats that could affect our protocol, based on the attacker's capabilities. We address how these threats are managed in Section 4.4. As to the assumptions we make about the attacker's strength based on the strength of the cryptography we use, we assume perfect cryptography.

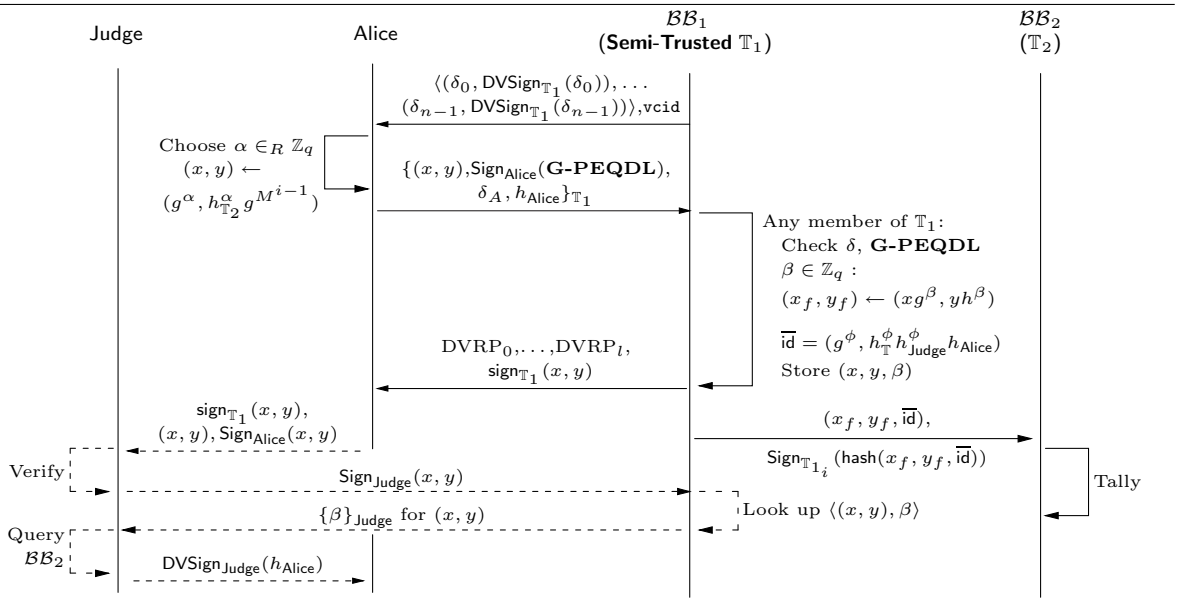
Note that in our protocol, the attacker can assume the role of any entity (except the Judge). He is able to corrupt up to $t-1$ talliers where collusion is required to decrypt messages (and t is the threshold size for that quorum). All channels are public, so the attacker can:

1. Read messages
2. Decrypt and read any message m, subject to having the correct decryption key s for an encrypted message $(g^{\alpha}, g^{\alpha s}m)$
3. Intercept messages
4. Inject bad ballots in the first phase, and spurious messages generally
5. Temporarily block messages (although we assume resilient channels for liveness)

4.4 Protocol

Our first voting protocol has four stages:

Figure 4.2 Our protocol. Dashed lines indicate a non-compulsory part of the protocol (complaints). Note that the first communication ($\mathbb{T}_1 \rightarrow$ Alice) is in-person.



Stage 1: Ballot Validity Tokens

The protocol begins with Alice registering *in person* to vote (this would be with $\mathbb{T}_1$). At this point, she receives a *random number of* values δ_i, which are generated *at the point of registration*. Each has a designated verifier signature $\mathsf{DVSign}_{\mathbb{T}_1 \to \mathsf{Alice}}(\delta_i)$ paired with it, which has been generated by a member of $\mathbb{T}_1$. However, only one of these signatures is valid (clearly, only the voter with the correct private key can verify this fact[1]). Alice hence receives a string

$$\langle (\delta_0, \mathsf{DVSign}_{\mathbb{T}_1}(\delta_0)), (\delta_1, \mathsf{DVSign}_{\mathbb{T}_1}(\delta_1)), \ldots, (\delta_{n-1}, \mathsf{DVSign}_{\mathbb{T}_1}(\delta_{n-1})) \rangle$$

The coercion-resistance Alice enjoys increases with n (i.e., the probability that the attacker can guess the correct δ value decreases with n).

Note that Alice would be able to generate designated verifier signatures at her liberty. Alice is able to calculate which of the signatures is valid for the value paired with it, and the tallier stores, on a private electoral roll (accessible only to $\mathbb{T}_1$) the valid δ value for Alice with her name. If Alice votes under coercion, since she received a random number of δ values, an observer cannot force her to use all values (she could conceal one or more, or arbitrarily insert values). Hence she simply votes using invalid δ values.

If she later votes without coercion[2], she sends the correct δ value with her vote as a 'proof' of validity. Upon checking for eligibility, the talliers simply check Alice's submitted δ value against the correct one stored on the private electoral roll . If she were to send a value for which the DV-Signature was incorrect when sent to her, this would alert the first-round talliers that her vote was made under coercion, which would alter their response to her. However, a coercer would not be able to distinguish a valid δ value from an invalid one, as he has no way of determining whether Alice herself made the designated verifier signature, or indeed whether the signature is valid.

[1] Note that in Figure 4.2, Alice also receives a token `vcid`. In practice, she would remember this value, and use it to demonstrate to her computer that she was not being coerced, rather than having to verify designated verifier signatures herself. To some degree, however, the real-world implementation of our work is outside of the scope of this thesis.

[2] We assume that Alice is able to vote unobserved, but she only needs to do this once.

Stage 2: Encrypted Vote Posting

As with other voting protocols using homomorphic encryption, we choose the form of the ballot in such a way that decryption of all ballots multiplied together leads to a simple tally of votes. A vote for the i^{th} candidate is given as $g^{M^{i-1}}$, where M is the maximum number of voters.

Voter Alice selects a value $\alpha \in_R \mathbb{Z}_q$, and encrypts her vote for candidate i using the public key of the *second round talliers*, to give $(x, y) = (g^{\alpha}, h_{\mathbb{T}_2}^{\alpha} g^{M^{i-1}})$. She groups this with the correct δ value δ_A, and her public key h_{Alice}. Finally, she calculates the Generalised Proof of Equality of Discrete Logarithms (see Section 3.1.3) for her ballot (x, y) to prove that the vote is of correct form, and produces a standard DSA signature on this. This tuple $\langle (x, y), \text{Sign}_{\text{Alice}}(\textbf{G-PEQDL}), \delta_A, h_{\text{Alice}} \rangle$ is encrypted with the public key of the first-round talliers, and posted to the first round bulletin board, $\mathcal{BB}_1$.

Stage 3: Validity Checking

Once Stage 2 is complete, any member $\mathbb{T}_{1_i}$ of $\mathbb{T}_1$ removes the first layer of encryption on each vote on the first-round bulletin board, supplying an appropriate proof of correctness if required by the authorities. That tallier then:

1. verifies that the vote is legitimate, by ensuring that the δ value given is the one stored with Alice's name on the private electoral roll[1]. Note that because the votes themselves are encrypted for $\mathbb{T}_2$, the first-round talliers cannot see *how* a voter votes — merely *that* a voter has attempted to vote.

2. verifies the G-PEQDL supplied with the ballot (x, y) to determine that Alice's vote is a single vote for a single valid candidate in the election

Once the validity of a ballot is assured, and any invalid ballots are disposed of, $\mathbb{T}_{1_i}$ re-encrypts (x, y) with a random factor β to give (x_f, y_f). That member also encrypts Alice's public key by doing the following:

[1] We presume that the private electoral roll is made inaccessible (or unconvincing) to coercers. We could accomplish this with designated verifier signatures.

- Select a random $\phi \in_R \mathbb{Z}_q$
- Using the joint public key for both sets of talliers $h_{\mathbb{T}}$, and the Judge's public key, form $\overline{\mathsf{id}} = (g^{\phi}, h_{\mathbb{T}}^{\phi} h_{\mathsf{Judge}}^{\phi} h_{\mathsf{Alice}})$.

The tallier then continues. He:

3. generates a signature on $\mathsf{hash}(x_f, y_f, \overline{\mathsf{id}})$, and concatenates this with $(x_f, y_f, \overline{\mathsf{id}})$ to form the final message string.

The tallier responsible for the re-encryption sends Alice a designated-verifier re-encryption proof (DVRP) that her vote has been included on the public bulletin board as (x_f, y_f), along with a number of other correct DVRPs, which are not valid for Alice (only she will be able to determine this), but *are* valid for other votes on $\mathcal{BB}_2$. Note that if Alice's sent δ value were invalid, the tallier would send Alice only DVRPs which were invalid for her (but still represented votes actually on $\mathcal{BB}_2$), meaning that an attacker could not determine whether her vote was invalid simply by observing messages received by Alice. As before, Alice would be free to insert seemingly valid DVRPs into the communication. The tallier also sends Alice a signature of her original vote, $\mathsf{sign}_{\mathbb{T}_1}(x, y)$.

The tallier will then personally store the values $\langle (x, y), \beta \rangle$, and mark on the private electoral roll that Alice has voted (for example, by adding a signature of her public key). This information will never be released, except to the Judge as proof that Alice's vote was counted. The tuple $\langle x_f, y_f, \overline{\mathsf{id}}, \mathsf{sign}_{\mathbb{T}_1}(\mathsf{hash}(x_f, y_f, \overline{\mathsf{id}})) \rangle$ is posted to the second-round talliers' bulletin board. Alice is able to check the second bulletin board to ensure her vote appears and the signature on it is valid, but cannot convince anyone else of this fact (nor can she decrypt the re-encrypted vote). Any entity can check that a vote on the bulletin board is valid by verifying the signature for the hash of that vote.

Stage 4: Tallying

Once all DVRPs have been sent to their respective voters, it is simple for the second-round talliers $\mathbb{T}_2$ to decrypt votes. First, each $\langle (x_f, y_f), \overline{\mathsf{id}} \rangle$ is checked against its signed hash. Those not matching

are ignored in tallying. A quorum of t talliers jointly decrypt a product

$$(X, Y) = (\prod_{j=1}^{l} x_{f_j}, \prod_{j=1}^{l} y_{f_j})$$

(without any single member having access to the private key, as discussed in Section 3.1.1), and then post the product to a publicly viewable place. The quorum threshold-decrypt the resulting tally, giving $g^{r_1 M^0 + r_2 M^1 + \ldots + r_L M^{L-1}}$, and $r_1, \ldots, r_L$ as the final tally, via the calculation of a single discrete logarithm. Note that any party can verify that any vote must have been correct, by comparing each published hash to the values given with it.

Anonymity Revocation

We have built into our protocol the ability to recover a voter's identity after the voting process is complete, but only with the co-operation of the Judge and a quorum of $\mathbb{T}$, the anonymity group. When Alice's vote is submitted to $\mathcal{BB}_2$, part of it is a token $\overline{\mathrm{id}} = (g^{\phi}, h_{\mathbb{T}}^{\phi} h_{\mathsf{Judge}}^{\phi} h_{\mathsf{Alice}})$ If, in the tallying phase of the protocol, any ballot is found to be illegal (or if, for any other reason, anonymity has to be revoked), a quorum of members of the *anonymity tallier group* $\mathbb{T}$ need to collude (note that the t_{id} value for this threshold decryption should be higher than the size of either $\mathbb{T}_1$ or $\mathbb{T}_2$).

$$\frac{h_{\mathbb{T}}^{\phi} h_{\mathsf{Judge}}^{\phi} h_{\mathsf{Alice}}}{g^{\phi s_{\mathbb{T}}}} = h_{\mathsf{Judge}}^{\phi} h_{\mathsf{Alice}}$$

The Judge must now be sent the token, with appropriate evidence justifying anonymity revocation. The Judge can then divide by $g^{\phi s_{\mathsf{Judge}}}$ to give the voter's identity.

Voter Complaints

A disadvantage of using designated-verifier re-encryption proofs is that Alice cannot prove the validity of the proof she receives from the first-round talliers that her vote has been re-encrypted as (x_f, y_f), which she may need to do if she cannot find her re-encrypted vote on $\mathcal{BB}_2$.

A solution we might adopt would be for Alice to receive a 1-out-of-L re-encryption proof

(Hirt and Sako, 2000), which is requested by Alice after all votes are posted to the board. However, such a proof is quite laborious and would allow an attacker to see that Alice's vote was counted. Instead, Alice sends her original (x, y) to the Judge, along with $\mathsf{sign}_{\mathbb{T}_1}(x, y)$ as proof that she did indeed submit that vote. The Judge requests the stored β from the first-round talliers, and can then use this to check that Alice's vote was counted. If Alice's vote is counted, the Judge sends her a designated verifier signature for her public key, h_{Alice}. Otherwise, she makes the designated verifier signature invalid. Only Alice can determine this fact, and can again insert valid signatures arbitrarily. If Alice's vote is shown to have not been counted, we could also allow her to collude with the Judge to submit a vote a second time—in this manner, if her vote is again not counted, the Judge can take further action.

4.5 Analysis

In this section, we provide a short list of the properties that this protocol satisfies. In Chapter 7, we go into considerably more detail by providing several formal models for the protocol, proving via the tool ProVerif that we satisfy many of the properties which we claim. Some properties (such as remote voting) cannot be tested formally, but it should be clear to the reader that these properties are achieved.

4.5.1 Coercing **Alice** by Selecting Her Keypair

We consider a potential attack to the protocol in which, rather than Alice using a private key of her choice, she is provided with a public key (or complete keypair) with which to vote. We consider first a situation in which the coercer provides alice with a public key ($h'_{\mathsf{Alice}} = g^{s'_{\mathsf{Alice}}}$), and uses his own copy of the corresponding private key s'_{Alice} in order to generate signatures where required, and decrypt designated verifier signatures and DVRPs sent to her. We can avoid this attack during in-person registration: we simply have Alice interactively prove knowledge of the secret key, thus ensuring that she possesses both secret and private keys.

In the case where the coercer provides Alice with a public *and* private key to use, the situation

is somewhat different. Foremost, we might argue that this allows the coercer to simulate Alice entirely, which contradicts our earlier discussion on the capabilities of the coercer, in Section 2.1.1. Nevertheless, if we assume that this could happen, then Alice would possess the required keypair with which to interactively prove knowledge of the private key, meaning that the coercer could read designated-verifier messages sent to her. We can take one of two approaches to mitigate this issue. The first is to simply decide that the attacker is *not permitted* to force keys in this manner (this is the approach that we adopt in this work). An effective way of enforcing this requirement is to have a fresh voter keypair generated for Alice, either at registration, or at some other point in time—receipt of a keypair need not be related to voting at all. However, in the latter case, we would require that $\mathbb{T}_1$ needs to be certain of the origins of Alice's public key: perhaps she needs to undergo an interactive proof knowledge of the private key, *and* the key itself is certified as having been produced for Alice. The former case, where Alice's keypair is generated for her freshly during registration, is more interesting. If the registrar generates Alice's keypair entirely, then we must assume that the registrar cannot be a coercer, else the coercer would have full knowledge of Alice's keypair anyway. A sensible alternative is to have Alice's keypair generated using two sources of randomness: one controlled by Alice, and the other controlled by another party—perhaps $\mathbb{T}_1$, or perhaps some other Certificate Authority, external to the protocol. We must again assume that the coercer is not able to control the registrar, if the registrar generates randomness for Alice's keypair, but this time, we need only be concerned if the coercer can *also* access to Alice's own source of randomness. In practice, many protocols of this type suffer from a similar weakness: if the coercer can, directly or otherwise, obtain Alice's secret data, then the coercer can simulate Alice entirely, making the whole effort futile. For this reason, as discussed earlier, we assume that the coercer *cannot* simulate the registrar in order to gain Alice's registration secrets.

The second approach which we might have adopted to securing Alice's registration is to *allow* the coercer to force public keys, but strengthen the way in which Alice is assured of which δ value—out of those she is sent in the first communication between $\mathbb{T}_1$ and her—is valid. Currently, we use designated verifier signatures on each δ value, where the single value which has

a 'correct' designated verifier signature is the 'valid' δ value. If we replace this communication with an interactive zero-knowledge proof of which δ value is correct, we then give Alice further ability to deceive the coercer: note that zero-knowledge proof transcripts can be simulated post facto, meaning that Alice is easily able to generate fake proofs to fool the coercer—*after* in-person registration has occurred—as to which δ is valid. A similar approach to proving credential validity was used by Benaloh and Tuinstra (1994), and later by Neff (2004) in the MarkPledge system.

Given the added complications of the second solution (in assurance that a given vote has been cast, and the added effort required by Alice in registration), we require that the coercer cannot force keypairs in this protocol.

4.5.2 Properties Satisfied by First Protocol

The protocol described in this chapter satisfies the properties listed below. We use the Dolev-Yao model and hence assume that the cryptographic operations presented in Chapter 3 are perfect; in other words the intruder is not able to break any of the these cryptographic algorithms but is able to intercept, change and delete all messages. We assume resilient channels to obtain liveness properties.

1. **Eligibility** Only eligible voters should be able to vote.

2. **Uniqueness** Only one vote per voter should be counted

3. **Receipt-Freeness** The voter should be given no information which can be used to demonstrate to a coercer how *or if* they have voted, *after* voting has occurred

4. **Coercion-Resistance** It should not be possible for a voter to prove how they voted or even if they are voting, even if they are able to interact with the coercer during voting

5. **Verifiability**

   (a) **Individual Verifiability** A voter should be able to verify that their vote has been counted correctly

(b) **Universal Verifiability** Any observer should be able to verify that all votes have been counted correctly

6. **Fairness** No-one can gain any information about the result of the tally until the end of the voting process and publication of votes

7. **Vote Privacy** Neither the authorities nor any other participant should be able to link any plaintext ballot to the voter having cast it, *unless* the protocol to revoke anonymity has been invoked

   (a) **Revocable Anonymity** It should be possible for an *authorised entity* (or collaboration of entities, for us) to reveal the identity of any *single* voter by linking his ballot to him.

8. **Remote Voting** Voters should not be restricted by physical location

It should be noted that even in the event that $\mathbb{T}_1$ were not trusted and became compromised, vote privacy, fairness, and individual verifiability (in so much that Alice can ensure her vote is counted), are still satisfied—these are not dependent on trusting $\mathbb{T}_1$. The fact that Alice uses the public key of $\mathbb{T}_2$ to encrypt her vote means that a corrupt $\mathbb{T}_1$ would have no access to it whatsoever. Receipt-freeness and coercion-resistance are satisfied in that Alice still cannot show *how* she votes.

The assumptions we make on $\mathbb{T}_1$ make it unnecessary to require assumptions made in other approaches on remote electronic voting, e.g. anonymous, often untappable channels (Sako and Kilian, 1995; Fan and Sun, 2008; Fujioka et al., 1993; Hirt, 2001; Cramer et al., 1996)[1], availability of a trusted Smart-Card or 'randomiser' to perform re-encryptions and proofs thereof (Lee et al., 2004; Hirt, 2001; Fan and Sun, 2008), or the assumption that the voter cannot be observed at all during voting. It should be noted that using a Smart-Card to re-encrypt instead of $\mathbb{T}_1$ would affect other properties, such as eligibility and remote voting.

[1]Note that the first stage of our protocol involves *in-person* registration, which one might class as an 'untappable channel'. In order to prevent Alice being simulated entirely by an attacker, this part of the protocol *must* be carried out by Alice alone. Unlike many other voting protocols, however, only registration must be done in this way.

4.6 Summary

In this chapter, we have introduced a protocol providing remote electronic voting with revocable anonymity. The protocol allows for simple and anonymous tallying of votes, whilst also permitting an authorised judge to request anonymity revocation on any ballot.

The protocol has a number of advantages: we achieve a novel method of providing coercion-resistance, even in the presence of a coercer; the protocol is the first to discuss revocable anonymity in electronic voting, as required by the UK; we achieve a voter- and universally-verifiable tally through the encryption and re-encryption methods which we use. However, it has some shortcomings. Firstly, the amount of trust required in the first set of talliers—though acceptable in a real-world scenario, we believe—is quite high, and something we we would like to reduce. Moreover, although this protocol is for a remote electronic voting scenario, it, like many other remote protocols, does not consider the security of the remote machine Alice votes from. This means, for example, that a rogue machine could claim to accept Alice's vote, and provide her with apparently correct validations of the proofs sent to her, but in fact vote on her behalf for another candidate. In this scenario, the 'weak link in the chain' becomes Alice's machine.

In the next chapter, we introduce another protocol, which harnesses the security guarantees provided by *trusted computing* and the TPM, in order to provide some assurances to the voter and the authorities as to the state of Alice's machine.

5 Using Trusted Computing[‡]

Chapter Overview

In this chapter, we build on the work discussed in Chapter 4, discussing a coercion-resistant electronic voting protocol which satisfies a number of properties previously considered contradictory. We introduce trusted computing as a method of ensuring the trustworthiness of remote voters, and provide an extension to our protocol allowing revocable anonymity. The protocol introduced in this chapter solves a number of the issues with the work presented in Chapter 4, including that of excessive trust in the first round of talliers. The protocol is a modified version of work presented at WISSec 2010 (Smart and Ritter, 2010), and is to be presented at Autonomic and Trusted Computing 2011 (Smart and Ritter, 2011).

We introduce the first practical work on a *remote* electronic voting protocol which uses *trusted computing* (specifically, the TPM and Direct Anonymous Attestation protocol). A number of

[‡]This chapter is an extended version of work presented at the Eighth International Conference on Autonomic and Trusted Computing (Smart and Ritter, 2011). An earlier version was presented at WISSec 2010.

existing works discuss the applicability of trusted computing and the TPM to electronic voting. We are the first to extend this to remote electronic voting whilst also providing a detailed protocol to do so, leading to several contributions:

- A remote voting protocol allowing authorities to be convinced of the state of the voter's machine, and allowing anonymity revocation via the TPM
- A protocol allowing Alice to remain anonymous, whilst satisfying her eligibility to vote via a novel use of the DAA protocol
- A novel method of allowing the voter to achieve coercion-resistance, whilst also achieving verifiability even in the *physical presence* of a coercer, such that one cannot determine *even if* a voter has voted (a notion we name *invisible absentee coercion-resistance*), through the use of designated verifier re-encryption proofs
- An extension to the protocol allowing a voter to be traced to her vote, should the legal need arise, but only with the co-operation of a judge.

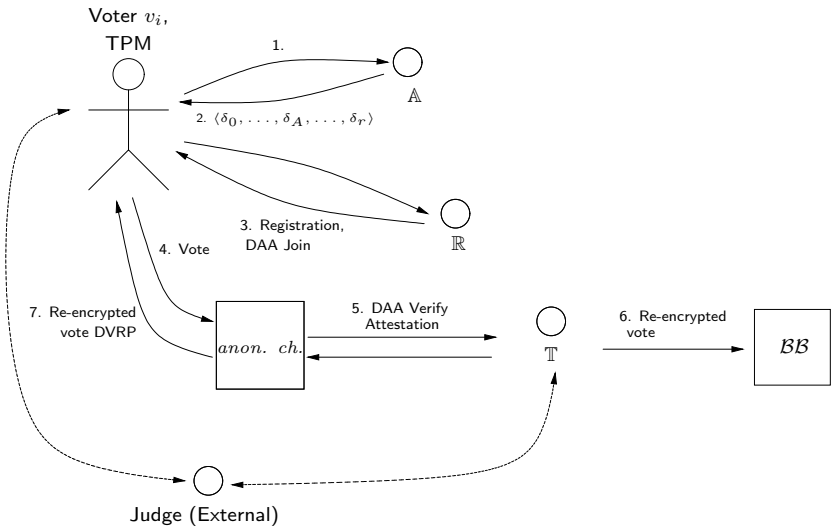
5.1 Chapter Structure

We have already discussed all of the primitives we use in Chapter 3. Presently, we introduce the schema for our second protocol, and then in Section 5.2, we give the participants, trust model and threat model for our work. In Section 5.3 we present our second protocol. We then provide a security analysis for the protocol in Section 5.4. We summarise the achievements of the protocol in Section 5.5.

5.1.1 Protocol Schema

We present a three-phase protocol, where voters do not need to synchronise between phases. In the first phase, our legitimate voter, Alice, registers *in person* to vote, and selects a random number of (paper) *validity cards* showing printed values δ_i, one of which (δ_A) she chooses at random. This

Figure 5.1 A schematic for our protocol. Alice begins by receiving validity tokens δ, in person. Dotted lines indicate optional communication



δ_A will denote her vote as non-coerced. In the next phase, she and her trusted platform module (TPM) execute the DAA *Join* protocol (Brickell et al., 2004) and receive a certificate proving her eligibility to vote (the certificate is split into three parts, divided between Alice and her TPM).

In the final phase, Alice and her TPM execute the DAA *Sign* protocol in order to complete her vote, which is sent as an ElGamal encryption with a proof of its validity. Voting authorities execute the DAA *Verify* protocol, after which Alice's vote is re-encrypted, and she receives back a designated verifier proof of that re-encryption, encrypted for her TPM. Should Alice need to, she can request assistance from the Judge if she cannot find her re-encrypted vote on the reported bulletin board, who may collaborate with her to cast a vote.

A simple schematic diagram of the protocol is given in Figure 5.1.

5.2 Protocol Model

5.2.1 Participants

Our protocol is modelled with four kinds of participants. All participants are able to communicate via a network, which is not untappable.

- **Voters.** The protocol allows M voters

$$v_i \in \{v_0, v_1, \ldots, v_{M-1}\}$$

to vote. Alice is an honest voter who wishes to vote anonymously. She can vote an unlimited number of times, but must be able to vote *once* unobserved. Voters' public keys are known to all participants.

- **Administrator.** The (in-person) administrator $\mathbb{A}$ is a single entity, responsible for ensuring that Alice receives a random number of paper *validity cards* containing validity tokens δ_j. We expand upon this in the next section. $\mathbb{A}$ is responsible for identifying Alice (say, via an ID card), but not for determining her eligibility to vote.

- **Registrar.** The registrar $\mathbb{R}$ is a single agent, possessing a secret key $s_{\mathbb{R}}$. Note that we assume a bottleneck will *not* occur here, but we could equally use a group of identical registrar agents to mitigate such a problem (though we would thereby, of course, increase the risk of data leakage).

  The registrar is responsible for ensuring, via the DAA *Join* protocol, that Alice is eligible to vote, and has not attempted to register already. The registrar will send Alice a voter group membership certificate, with which she can prove to the talliers that her vote is permitted.

- **Talliers.** The talliers, $\mathbb{T} = \{\mathbb{T}_1, \ldots, \mathbb{T}_n\}$, are a group of agents (disjoint from $\mathbb{R}$) who authorise the addition of each submitted ballot to the *bulletin board*, $\mathcal{BB}$, via the DAA *sign* and *verify* protocols. Each tallier has a copy of a secret key $s_{\mathbb{T}}$, with which he determines the validity of votes, and a *share* of a secret key $s_{\mathbb{T}_v}$, with which he collaborates with a quorum of $\mathbb{T}$ in order to decrypt the end tally, once the election is finished. These keys are unrelated—we use them to ensure that no single teller has access to an individual vote. $\mathbb{T}$ are also responsible for re-encrypting valid votes, and sending proof of this to Alice.

5.2.2 Trust Model

We make the following assumptions in our protocol:

1. The TPM and the manufacturers of the TPM (the root of trust), are trusted to behave as intended by the protocol

2. All parties trust that $\mathbb{T}$ will not reveal the link between a ballot (x, y) and its re-encryption (x_f, y_f)

3. All voters trust that the validity of any given δ value will not be revealed by $\mathbb{A}$, *except* to members of $\mathbb{T}$ via a designated verifier signature

4. All parties trust that each voter will only be permitted to submit *one* validity card to the secured box for each election

5. All parties trust that $\mathbb{R}$ will not issue group membership certificates to ineligible voters, and will only do so once for eligible voters

6. All participants trust that the Judge will only authorise revocation of anonymity in appropriate circumstances

7. Alice trusts the Judge to honestly state whether votes have been counted

5.2.3 Threat Model

We now consider the potential threats that could affect our protocol, based on the attacker's capabilities. We address how these threats are managed in Section 5.3. Note that, as mentioned earlier, we assume perfect cryptography.

In our protocol, the attacker can assume the role of any entity, except the Judge or $\mathbb{A}$. He is able to corrupt up to $t - 1$ talliers where collusion is required to decrypt messages (and t is the threshold size for that quorum). All channels are public (the channel between Alice and the talliers is tappable, but anonymous), so the attacker can:

1. Read messages
2. Decrypt and read any message m, subject to having the correct decryption key s for an encrypted message $(g^{\alpha}, g^{\alpha s}m)$
3. Intercept messages
4. Inject bad ballots in the voting phase, and spurious messages generally
5. Temporarily block messages (although we assume resilient channels for liveness)

5.3 Protocol

Our protocol has three stages. Diagrams of these are given in Figures 5.2, 5.3 and 5.4:

In-Person Registration (Fig 5.2) In order to begin voting, Alice first has to apply *in person* to vote, with the administrator $\mathbb{A}$. This can be at any point before the election. It is at this stage that her identity is confirmed. Once her identity is confirmed, Alice is told to select a number, r, of *validity cards* from a box. r is generated randomly by $\mathbb{A}$ when Alice's identity is confirmed, and she is observed selecting (at least) r face-down cards from the box. Alice is free to select a further, arbitrary, number of cards. These cards are simply pieces of paper with a perforation down the middle, and the same value $\delta_j : j \in \{0, \ldots, r-1\}$ printed on each side. Alice selects (mentally) one of the cards, whose δ value, δ_A, will denote her intended vote. She separates the card along the perforation, and places half of it into a secure box, retaining the other half. The bin must be designed to accept only one card per voter. Designing such a bin is an interesting challenge. One solution is to have the administrator $\mathbb{A}$ 'reset' the box for each voter after having confirmed their identity. Alice then selects a random number of cards, and places one in the box. The box weighs the card before accepting it, preventing 'stuffing', and then locks itself until reset again. The physical implementation of such a bin is very much outside of the scope of this work, however, and so we do not discuss it further.

With the remaining cards, Alice separates each card and places one half of each card into a shredder. Again, we should ensure that Alice destroys half of each validity card that she has not

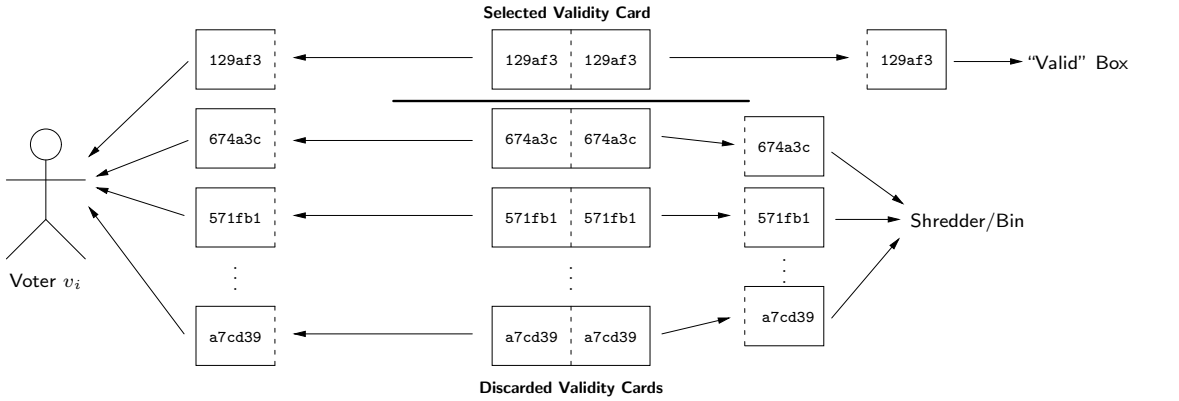
chosen to denote her intended vote (as is the case with protocols such as Prêt-à-Voter, discussed in Section 2.2.4.1).

Alice leaves in-person registration with several halves of validity cards. She has a mental note of which is valid (and could, in fact, discard or hide that one), but cannot prove which is valid to any observer. As she took a random number of cards, an observer cannot force her to vote once with every card she selected. Note that only $\mathbb{A}$ has access to the secure box, and that the voter has no way to prove how many cards she selected.

As an interesting aside, the reader may note that there is actually no side-effect of Alice *not* destroying half of each unwanted validity card in this way. Let us assume that Alice destroys none of the card halves. She therefore leaves the polling station with several whole cards, and one half of a card (whose other half is in $\mathbb{A}$'s secure box). No coercer is able to determine if the total number of cards Alice has is indeed the total number she was allocated, plus the further number she chose to take: the most that can be determined is that the 'whole' cards do *not* show δ_A. Indeed, Alice may have thrown away, or otherwise concealed, the card which does show the δ_A value. As we will discuss later, the proofs which Alice receives from the talliers as to her vote being tallied are firstly encrypted for her TPM only, and secondly only readable by Alice, meaning that the coercer can never gain information about the validity of any δ value.

A related attack which we consider is that of "chain registration" (cf. chain voting): the coercer has access to two voters, under the assumption that voters now *do not* destroy the halves of their unwanted cards. With the first voter, he asks for all of the 'complete' cards (i.e., those which have not been divided). Now, he gives those cards to the next voter, asking her to select the δ value on one of those to vote with, instead of any that she will receive as part of the registration process. The question then follows: can the coercer determine that the voter has indeed voted the way he requested, having knowledge of which δ value she chose to denote a valid vote? Again, this attack cannot succeed. As we will discuss later, when Alice votes, she sends her single δ value as part of her vote. The value is checked for validity, and the validity of a given value is denoted by a valid designated verifier signature on that value, which the coercer will not be able to verify. When a vote is recorded on the tally, it is re-encrypted randomly. Proofs of this

Figure 5.2 In-Person Registration

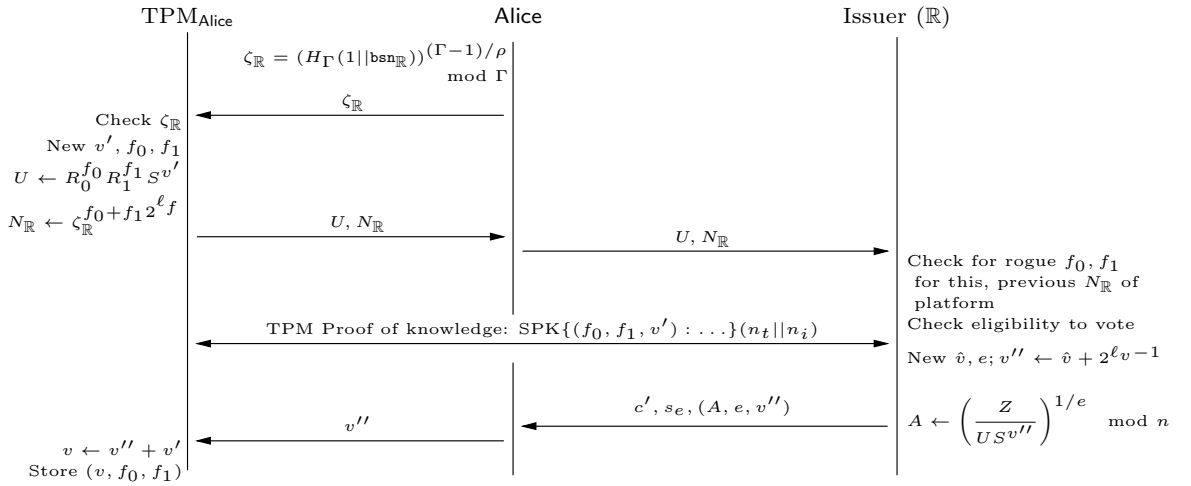


re-encryption are verifiable only by Alice, and are encrypted using a PCR-bound TPM key to begin with. This means that only Alice, working with her TPM, can verify that any vote appears on the tally. Though the effect of this is that Alice does not in fact need to destroy the unwanted validity card halves, in the current version of our protocol, to minimise complexity, we enforce that she should do so.

We note that our approach to voter registration is unconventional for a remote electronic voting scheme. However, it removes the unrealistic requirement for an untappable channel to the administrator (like that suggested by Clarkson et al. (2008)), and considerably reduces the trust we need to place in $\mathbb{A}$ (he now only knows which δ values are valid, not for whom, so we need only ensure that he does not release this information).

Join (Fig 5.3) Alice and her TPM, $\text{TPM}_{\text{Alice}}$, execute the TPM *Join* protocol: this is as with the DAA *Join* protocol (Brickell et al., 2004), which we discuss in depth in Section 3.2.2.3. Alice first forms a value $\zeta_{\mathbb{R}}$, using the *basename* of the registrar, and sends this to her TPM. The TPM checks the validity of the value, selects random values v', f_0, f_1 and forms a commitment to them, U, and a pseudonym $N_{\mathbb{R}}$ with which to allow the registrar to identify Alice. $U, N_{\mathbb{R}}$ are sent back to Alice, who sends them to $\mathbb{R}$. The communication channel with $\mathbb{R}$ does not need to be anonymous. We however adopt the requirement of (Brickell et al., 2004) that the channel must be 'authentic' between $\text{TPM}_{\text{Alice}}$ and $\mathbb{R}$: i.e., the registrar must be sure that it is communicating with the correct TPM. Such authenticity can be achieved using the TPM's *endorsement key* (EK) for initial communications (Brickell et al., 2004).

Figure 5.3 The Join protocol, where Alice registers to vote.



ℝ checks to see whether Alice has already applied to vote, or whether the TPM she is voting from has been designated rogue. If not, $\text{TPM}_{\text{Alice}}$ and ℝ engage in a signature proof of knowledge protocol, with the TPM as the prover. The TPM proves knowledge of f_0,f_1 and v', the blinding factor in U.

Once this is complete, ℝ generates Alice's membership certificate:

$$(A, e, v'') : \quad A \leftarrow \left(\frac{Z}{US^{v''}}\right)^{1/e} \mod n$$

We refer to Figure 5.3, Section 3.2.2 and (Brickell et al., 2004) for more detail. This certificate demonstrates the voter with pseudonym $N_{\mathbb{R}}$'s eligibility to vote, without revealing her identity.

Note that we do *not* need anonymity in the first part of the protocol—in fact, since ℝ *needs* to identify Alice on the electoral register, this part of the protocol *could not* be anonymous. Instead, we need to ensure that there is no way to link the information gained by Alice in the *Join* protocol with the information she imparts in the *Vote* protocol, by using two different pseudonyms to identify Alice in the registration and voting phases, and an anonymous channel in the voting phase.

The certificate is sent to Alice. She then stores most of it, and forwards v'' to $\text{TPM}_{\text{Alice}}$, which can calculate $v \leftarrow v' + v''$, and store (v,f_0,f_1).

Voting (Fig 5.4) The protocol by which Alice and her TPM vote is shown in Figure 5.4. If

we assume that Alice can be tracked by an attacker with a global view of the network (and thus, the ability to see the IP address Alice votes from), then we must use an anonymous channel to preserve Alice's coercion-resistance and privacy. Although using any form of anonymous channel is undesirable, we do gain the valuable property that the machine Alice votes from can not be traced. Voting proceeds as follows:

First, we begin with an execution of the DAA *Sign* protocol (denoted as such in Figure 5.4). Alice and her TPM form tokens with which she can prove possession of a credential supplied by $\mathbb{R}$. First, as with the *Join* protocol, Alice forms a value $\zeta_{\mathbb{T}}$, and sends this to $\text{TPM}_{\text{Alice}}$. Meanwhile, she generates two tokens T_1, T_2, which are used to demonstrate to $\mathbb{T}$ that she possesses a certificate making her a member of the 'voters' group, without revealing the certificate itself (Brickell et al., 2004). Next, $\text{TPM}_{\text{Alice}}$ forms a pseudonym $N_{\mathbb{T}}$ with which Alice can vote (note that $N_{\mathbb{T}} \neq N_{\mathbb{R}}$), then sends this to Alice. She forwards $N_{\mathbb{T}}$ to $\mathbb{T}$, and then $\mathbb{T}$, Alice and $\text{TPM}_{\text{Alice}}$ engage in a signature proof of knowledge, with Alice and her TPM producing a signature proving that T_1 and T_2 commit to a certificate, and "[her pseudonym $N_{\mathbb{T}}$] was computed using the secret key going with that certificate"(Brickell et al., 2004). Alice's TPM generates an *attestation identity key* (Brickell et al., 2004) $\text{AIK}_{\text{Alice}}$ which is sent to $\mathbb{T}$ as part of this signature, and will be used to prove authenticity of later messages. Note that this AIK is not linkable to Alice in any way, and the communication with $\mathbb{T}$, being over an anonymous channel, is similarly unlinkable (Brickell et al., 2004).

With the *Sign* protocol complete, $\mathbb{T}$ can then query Alice's TPM as to the state of her machine. To do this, any member $\mathbb{T}_i$ of $\mathbb{T}$ begins an *encrypted transport session* between itself and Alice's TPM directly (note that Alice does not see the result of any transactions that occur here). $\mathbb{T}_i$ selects a challenge nonce c_v, and requests a hash of the current state of the TPM's registers, using the command `TPM_Quote`, and including the challenge. The TPM responds with the appropriate data. If $\mathbb{T}_i$ is satisfied that the machine is in the correct state, it requests that the TPM create a new keypair, bound to the correct TPM register (PCR) states. This means that, when a decryption is needed using this key, it can only occur if the TPM's PCRs are in the correct state. We denote

the *handle* of this key as k_A, and note that the key is asymmetric, the private part being accessible only to the TPM.

Next, Alice generates a fresh ElGamal keypair, $(s_v, h_v = g^{s_v})$. She then sends a message `votetoken` to $\mathbb{T}$. `votetoken` contains Alice's vote, in the form of an exponential ElGamal encryption $(x, y) = (g^{\alpha}, h_{\mathbb{T}_t}^{\alpha} g^{M^{i-1}})$, where she is selecting the i^{th} candidate, her chosen δ_A value (should she be voting according to her own wishes) or any other δ value (if she is being coerced), the public part of the aforementioned key h_v, and the **G-PEQDL** proof that her vote is for one valid candidate only. The tallier $\mathbb{T}_k$ that receives Alice's vote now checks whether it was sent under coercion. To do this, he sends $\delta, \mathsf{sign}_{\mathbb{T}}(\delta)$ to $\mathbb{A}$. $\mathbb{A}$ checks whether the δ value received is in the secure box, and if so, sends a correct designated verifier signature of the value, $\mathsf{DVSign}_{\mathbb{A}\to\mathbb{T}_k}(\delta)$. If the δ value is not found in the box (meaning Alice sent a vote under coercion), an incorrect designated verifier signature is returned to $\mathbb{T}$. Again, only $\mathbb{T}_k$ can determine this, and cannot prove this fact to an observer.

Once Alice's vote is determined to be non-coerced, her G-PEQDL proof is checked by $\mathbb{T}_k$. If this is invalid, her vote is discarded. If the G-PEQDL is correct, Alice's vote is re-encrypted using a re-encryption factor $\beta \in_R \mathbb{Z}_q$. If her vote was not coerced, Alice is sent a tuple of designated verifier proofs of re-encryption (DVRPs), produced using the public key h_v Alice generated earlier. One of these is valid for Alice's re-encrypted vote; the others are valid for other votes already on the bulletin board[1]. Each DVRP is separately encrypted using the public part of the wrap key k_A which Alice's TPM generated. This means that Alice is free to generate re-encryption proofs herself (the nature of the proof is such that the entity for whom the proof is designated can use her private key—s_v in this case—to generate further DVRPs), to fool coercers.

The re-encrypted (x_f, y_f) is sent to a threshold of talliers in $\mathbb{T}$, along with the re-encryption factor and the G-PEQDL proof. If that threshold agree, they jointly generate a signature on (x_f, y_f), and the vote and its signature are placed on the bulletin board.

If Alice's vote *was* coerced, Alice is sent several DVRPs as before. However, this time they are all valid for votes on the bulletin board that are *not* Alice's.

[1] Vote submissions are batched so that there are always enough votes on the bulletin board to do this.

Note that the DVRPs Alice receives use a key which she freshly generated (to prevent her being identified). Each DVRP is encrypted with a key for which only the TPM has the private part. As a consequence, Alice needs to load the correct key into the TPM (using `TPM_LoadKey2`), and then requests the TPM to decrypt each DVRP ciphertext, using `TPM_UnSeal`.

At this point, it should be noted that the keypair k_A generated by the TPM was bound to a certain set of PCR states. If this set of states is not in place at the time of DVRP decryption with `TPM_UnSeal`, decryption cannot occur. This ensures not only that Alice still uses the same TPM, but also that no rogue software is executed after Alice casts her vote.

Alice can then check to see if any one of the DVRPs represent valid re-encryptions, checking the bulletin board. Note that *every* re-encryption will be on the bulletin board, but only Alice can be convinced that any vote is hers. If she does not find her vote, Alice may contact the Judge, who will contact $\mathbb{T}$. The Judge may further allow Alice to vote again, under his supervision.

When voting is complete, votes are tallied by $\mathbb{T}$:

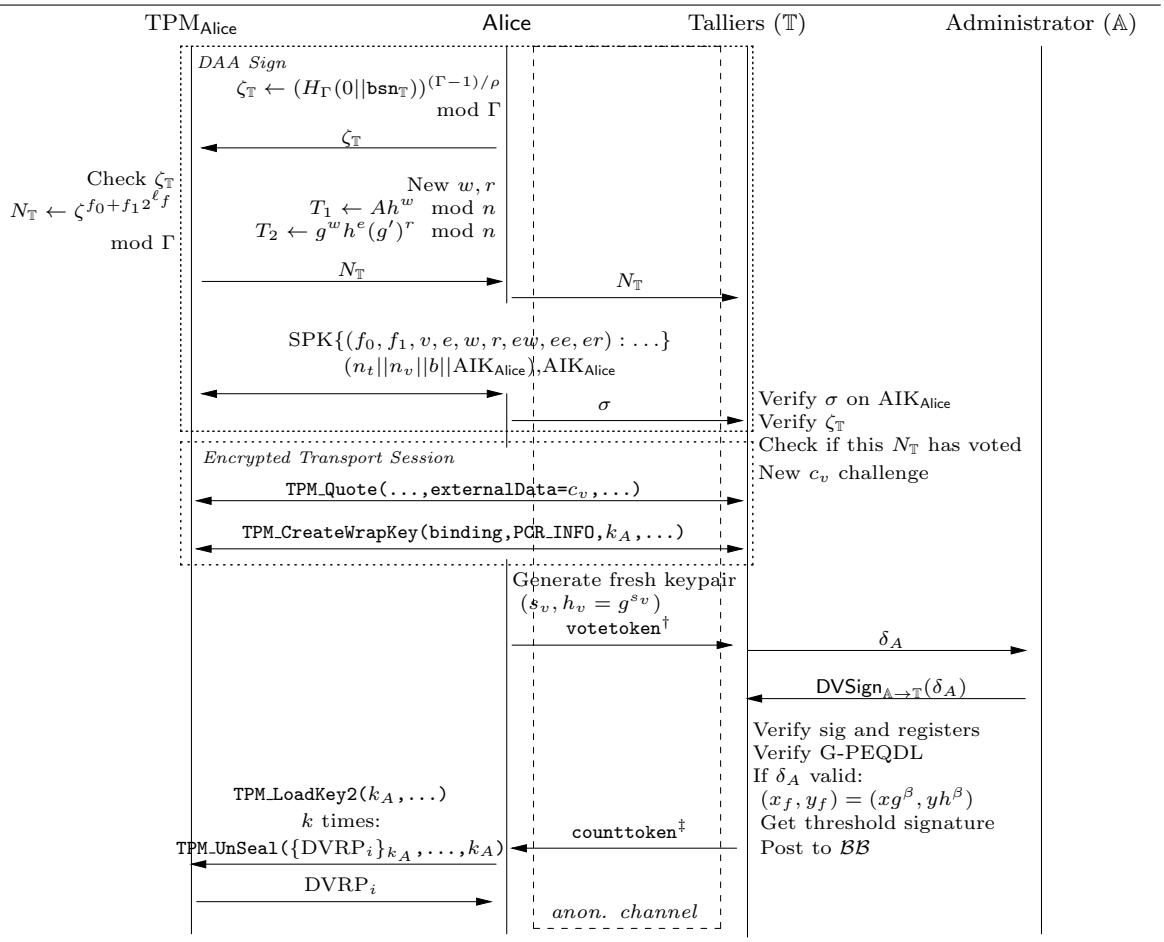
$$(X, Y) = (\prod_{j=1}^{l} x_{f_j}, \prod_{j=1}^{l} y_{f_j})$$

This product is calculable by any observer. The final tally is calculated by a quorum (size t) of $\mathbb{T}$ colluding to decrypt this product, giving $g^{r_1 M^0 + r_2 M^1 + \ldots + r_L M^{L-1}}$, and $r_1, \ldots, r_L$ as the final tally. Note that since every vote is threshold-signed on the bulletin board, observers are convinced that every vote is genuine.

Anonymity Revocation (Fig 5.5) The changes that we make to the protocol in Figure 5.4 in order to provide revocable anonymity are quite simple. We begin with a small change to the registration protocol. Once the DAA *Join* part of the protocol is complete, the registrar $\mathbb{R}$ sends Alice an encryption of her ID with the Judge's public key, $\overline{\mathsf{id}} = \{\mathsf{id}\}_{\mathsf{Judge}}$. $\mathbb{R}$ also sends a signature of this encryption, $\mathsf{Sign}_{\mathbb{R}}(\overline{\mathsf{id}})$ to Alice.

The voting protocol completes the DAA *Sign* protocol as before. Alice then sends the encryption and signature thereof to $\mathbb{T}$, who verify the signature and store the ciphertext. She then extends a TPM PCR with the value of $\overline{\mathsf{id}}$ using `TPM_Extend` (this is equivalent to hashing the current value of the chosen register, concatenated with $\overline{\mathsf{id}}$). $\mathbb{T}$ can ensure Alice has done this, by

Figure 5.4 The Voting Protocol `vote1` (without revocable anonymity). Terms are explained in Section 3.2.2.

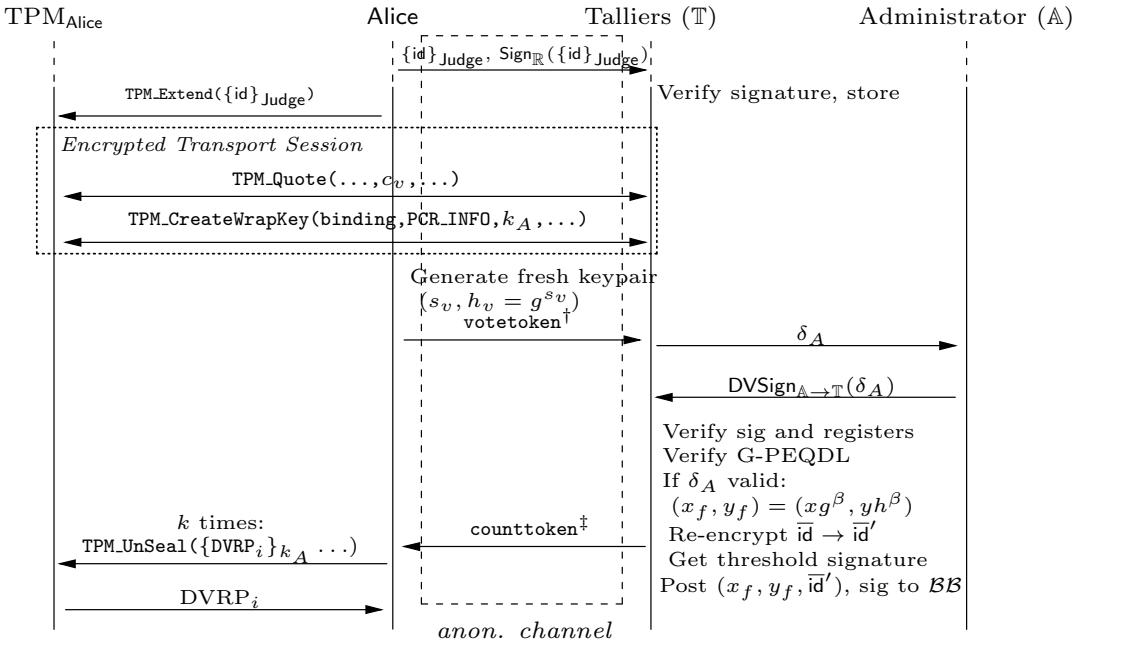


† : `votetoken` $= \langle (x,y) = (g^{\alpha}, h_{\mathbb{T}_v}^{\alpha} g^{M^{i-1}}), \mathbf{G-PEQDL}, h_v, \delta_A \rangle$
‡ : `counttoken` $= \langle k_A, \{\text{DVRP}_0\}_{k_A}, \ldots, \{\text{DVRP}_k\}_{k_A} \rangle$, one valid for h_v if δ_A correct

ensuring that the value received from `TPM_Quote` is that which would be expected for a correct machine state *concatenated with* the encrypted ID value.

Voting then proceeds as normal: Alice's identity is re-encrypted by $\mathbb{T}$ and printed on the bulletin board next to her vote. Should revocation be required, a member $\mathbb{T}_k$ of $\mathbb{T}$ sends the tuple $\overline{\mathsf{id}}$ to the Judge, along with appropriate evidence justifying revocation. The Judge is then free to revoke Alice's anonymity and take further action against her. Note that in order to preserve Alice's anonymity, we add a trust requirement that $\mathbb{R}$ does not collude with $\mathbb{T}$ to reveal Alice's identity, and always provides the correct identity for a voter (since $\mathbb{R}$ is trusted to perform the DAA Join protocol correctly, this is not a large increase in trust). Note that Alice could later contact the Judge to determine whether her anonymity had been revoked or not. This does not, to us, constitute full auditability, as Alice needs to contact a third party to audit her vote. We discuss approaches to achieving auditable revocable anonymity in the next chapter.

Figure 5.5 Changes to the Voting Protocol `vote2` (with revocable anonymity)



† : votetoken $= \langle (x,y) = (g^{\alpha}, h_{\mathbb{T}_v}^{\alpha} g^{M^{i-1}}), \mathbf{G-PEQDL}, \delta_A, h_v \rangle$

5.3.1 Choosing Values for Validity Cards

One might consider what form the δ values printed on the validity cards actually take. In order to minimise the memorisation demands placed on the voter, the values must be simple (or memorable) enough to be 'human-friendly'. At the same time, they must be complex enough that brute-force guessing attacks would be ineffective. In the version of the protocol presented here, each δ value is randomly chosen from some finite set of integers. Equally, however, we could choose from a set of dictionary words, or even—with some caveats—allow the voter to select the values on the cards herself. We would prefer to avoid this option, as humans are notoriously bad at making random decisions, and Alice could be forced to choose certain values by a coercer. That said, she could easily comply with such a demand, and then insert her own chosen value, memorising it, submitting it as her chosen δ_A, and destroying or hiding the counterpart card.

5.4 Analysis

The following properties are those satisfied by this protocol. These properties are what we consider to be the most important properties in electronic voting. For each property, we give an informal sketch of security proofs for the protocol. We assume the correctness of various cryptographic primitives and assumptions (discussed in Chapter 3)—note that although these primitives assume the provable security model, we work in the formal model, thus assuming that cryptography is perfect. We make no further assumptions about, or changes to, the primitives that we use.

Property 1 (Correctness and Eligibility). *Only eligible voters should be able to vote. Further, there is no trace of the protocol resulting in a successfully counted vote, from* Alice*, for candidate i, that did not begin with* Alice *voting for i, and there is only* one *trace that* did *begin with* Alice *voting for i*

To prove this requirement, we need to demonstrate that there is no way any two or more parties can collude to defraud Alice or the authorities. Nor is it possible for Alice to collude with another voter or coercer. We consider collusions between the parties shown in Table 5.1, and discuss them below. Bob represents any other voter, or coercer.

i. *Alice and any other voter or coercer (Bob).* We consider an attack in which Alice colludes with an attacker to attempt to vote several times, or claim that her vote was not counted.

When Alice attempts to vote, part of her voting token is our **G-PEQDL** proof that for her vote (x, y), $y \in \{h_{\mathbb{T}_\nu}^{\alpha} g^{M^0}, \ldots, h_{\mathbb{T}_\nu}^{\alpha} g^{M^{L-1}}\}$ where L is the number of candidates. For any vote to be accepted by $\mathbb{T}$, this proof must hold. Hence, Alice cannot vote for more than one candidate in any one vote casting.

Further, she cannot vote more than once—we have from Trust Assumption 5 that $\mathbb{R}$ will not permit Alice to register more than once, and from the protocol that a quorum of $\mathbb{T}$ would have to collude to permit two votes from the same voter. Alice is also unable to claim that her vote was not counted—the talliers are always able to show which vote was re-encrypted to appear on the bulletin board, if requested to do so by a judge. Further, the Judge can request that the talliers produce DVRPs using her public key, instead. The Judge is then able to verify that Alice's vote has appeared on the bulletin board.

ii. *Alice and the Administrator.* We consider an attack in which Alice and $\mathbb{A}$ collude to provide her with multiple valid δ tokens. Such an attack violates Trust Assumption 4, and so is not considered further.

iii. *Alice and the Registrar.* We consider an attack in which $\mathbb{R}$ allows Alice to register more than once, or to register if she is not eligible. Such an attack violates Trust Assumption 5, and so is not considered further. In the revocable anonymity protocol `vote2`, $\mathbb{R}$ may send Alice a different identity. This violates the trust assumption which we added to this protocol.

iv. *Alice and one Tallier.* We consider an attack in which Alice colludes with a second round tallier to request that her vote is altered. Since any modifications to votes require the

Table 5.1 Possible Collusions in our Second Protocol

	Bob	$\mathbb{A}$	$\mathbb{R}$	$\mathbb{T}_i$
Alice	i	ii	iii	iv
$\mathbb{R}$	v	vi	vii	viii
$\mathbb{T}_{j\neq i}$	ix	x	xi	xii

agreement of a quorum of $t < n$ members of $\mathbb{T}$, this attack is not possible. Further, modifications after posting to the bulletin board are not possible, since each re-encrypted vote is posted with its signed hash to the board.

v. *Registrar and a Coercer.* In this attack, the registrar would collude with a coercer to provide Alice with more than one certificate with which to vote, or to provide her with an invalid certificate. The first attack violates Trust Assumption 5, and the second would immediately be detected by Alice and her TPM.

vi. *Registrar and the Administrator.* The Administrator is only responsible for issuing voters with validity cards, something which has nothing to do with registration. As a result, there is no valid attack here.

vii. *Registrar and Registrar.* As there is only one registrar, this attack is not possible. It would also not be possible for $\mathbb{R}$ on his own to affect the election: we have already stated that he cannot collude with Alice or a coercer, and attempting to provide invalid certificates to voters would be detected immediately by the voters themselves, who could complain to a Judge.

viii. *Registrar and one Tallier.* In this attack, $\mathbb{R}$ would generate an invalid certificate for Alice and collude with $\mathbb{T}$ such that the certificate was accepted. Since the values generated by $\mathbb{R}$ for the certificate are not directly seen by $\mathbb{T}$ this attack would fail. It also violates Trust Assumption 5. $\mathbb{R}$ could not force $\mathbb{T}_i$ to reject a certificate, for the same reason.

Note that in the protocol `vote2`, including revocable anonymity, we add a trust assumption to ensure that Alice's identity will not be revealed to the talliers by $\mathbb{R}$. This nullifies any collusion between the registrar and talliers.

ix. *A Coercer and one Tallier.* A coercer may attempt to force a tallier to accept a fraudulent vote, or to modify one. Both of these would require a signature from a quorum of $\mathbb{T}$, which means that collusion with a single tallier would be ineffective.

x. *One Tallier and the Administrator.* In this attack, a tallier would collude with the administrator (and possibly a coercer) to reveal a list of valid δ values, and the names associated with them. We have from the protocol that the administrator never sees who selected each δ value. Further, the validity of any δ value will not be revealed by $\mathbb{A}$, except via a designated verifier signature (meaning the recipient of the signature cannot prove the validity of any δ value).

xi. *One Tallier and the Registrar.* See (viii).

xii. *Any two Talliers.* A quorum of $t < n$ talliers is required to effect any change in the tally. Hence, no attack is possible here.

Collusion of more than two parties. Note that it follows from the discussion above that collusion of more than two parties to defraud Alice or the authorities is similarly ineffective. Any attempt to alter Alice's vote, or to allow her to vote multiple times at once, would require the cooperation of a quorum of $\mathbb{T}$—something which, as with all voting schemes using threshold cryptography, we assume is unlikely. We trust that $\mathbb{R}$ will not allow Alice to register more than once, and we also note that Alice working with any number of coercers would have no effect (subject to the assumptions above): i.e., any collusion involving more than two parties will reduce to the same collusions described above, which either require a quorum of cooperating talliers, or already break one of our trust assumptions.

Property 2 (Uniqueness)**.** *Only one vote per voter should be counted* We have from the protocol that, during in-person registration, Alice can only place one validity card into the secure box. Trust Assumption 4 states that Alice can only receive one such card. As such, given that a particular δ value will only be accepted once when the Talliers come to group-sign a re-encrypted vote, Alice can only vote once per election. Similarly, she is given only one group membership certificate in the *join* phase, and so cannot vote twice there, either—we trust that Alice's TPM will not allow her to generate more than one pseudonym.

Finally, when Alice comes to vote, she submits a **G-PEQDL** proof that her vote (x, y) is such

that $\gamma \in \{h_{\mathbb{T}_t}^{\alpha} g^{M^0}, \ldots, h_{\mathbb{T}_t}^{\alpha} g^{M^{L-1}}\}$ where L is the number of candidates. She can therefore not vote for more than one candidate in an election.

Property 3 (Coercion-Resistance). *It should not be possible for a voter to prove how they voted or even if they are voting, even if they are able to interact with the coercer during voting* We have from the protocol that Alice receives a random number of *validity cards* at in-person registration. She halves each card, putting half of only one card into a secure box. She discards the remaining halves, and remembers which was her valid number (she is advised to discard the half of her 'valid' card that isn't in the box).

Alice has no way to prove to a coercer that the δ value she submits her vote with is valid. Since she may have thrown away the correct card, the coercer can force her to vote using all of the values she holds. However, the proofs that Alice receives back are intended for a fresh keypair which Alice generated (meaning she can generate such proofs herself, and a coercer cannot determine this fact). As such, there is no way for an in-person observer to force Alice to vote in any particular way, as she can fool that coercer into believing that votes on the bulletin board represent re-encryptions of a vote *she* cast.

Property 3.1 (Invisible Absentee Coercion-Resistance). *It should not be possible for a voter to show* if *they have voted, even in the presence of a physical coercer* We extend our definition of Coercion Resistance to include *invisible absenteeism*—that is, even in a remote voting system, as long as Alice can vote once unobserved, a coercer physically standing behind her cannot tell if she has successfully voted or not.

This property is satisfied by our system because of the nature of the proofs Alice receives when she votes. She receives in return, over an anonymous channel, only a tuple of designated verifier re-encryption proofs, which only she can interpret. Alice can then observe the bulletin board without an in-person observer knowing whether she has genuinely found her vote, or is pretending to look for (x_f, y_f) pairs which she put into a DVRP generated by herself.

Property 4 (Receipt Freeness). *The voter should be given no information which can be used to demonstrate to a coercer how* or if *they have voted,* after *voting has occurred* Receipt-freeness is strictly a sub-property

of coercion resistance. The only proof sent to Alice of her vote is in the form of designated verifier re-encryption proofs, which can only be read by her. As a result, she cannot demonstrate to any observer how she votes, so receives no receipt.

Property 5 (Individual Verifiability)**.** *A voter should be able to verify that their vote has been counted correctly* When Alice submits her vote (x, y) (over an anonymous channel), it is encrypted using a random seed α. If valid, the vote is re-encrypted using a random seed β, giving (x_f, y_f), and it, with a threshold signature of its hash, is posted to the bulletin board.

Alice receives back from the talliers a random number of *designated verifier proofs of re-encryption*, which prove, to a designated verifier, how a re-encryption occurred. *All of these DVRPs are valid* for votes that are on the bulletin board, and *one of them* is valid for Alice's vote, if and only if the δ value she supplied with her vote was valid. The verifier of the DVRP is Alice, using a key which she freshly generated before voting. Since Alice can observe whether each DVRP is correct, she can ascertain whether a DVRP which is correct *for her* represents a ballot which is on the bulletin board.

Note that all of the (x_f, y_f) pairs Alice receives are listed on the bulletin board: only Alice can determine if her vote is one of them. If not, she can contact the Judge, who will allow her to vote again under supervision.

Property 6 (Universal Verifiability)**.** *Any observer should be able to verify that all votes have been counted correctly* We have from the protocol that every vote posted to the bulletin board is threshold-signed by a quorum of $\mathbb{T}$. Hence, invalid votes are not posted to the bulletin board. The talliers announce the product (X, Y) of all votes. As all votes are shown on the bulletin board, any observer is able to calculate this product for themselves, and also to verify that any individual vote was authorised by a quorum of talliers.

Property 7 (Fairness)**.** *No-one can gain any information about the tally until the end of the voting process* The phased structure of the protocol implies that it is fair. Observation of any single vote is not possible at the talliers' end, since a quorum of $\mathbb{T}$ have to agree to decrypt a vote. Similarly, the product of all votes, leading to information on the full tally, will not be decrypted until a quorum of $\mathbb{T}$ agree, which they will not until all votes are entered.

Note that it is also impossible to break fairness from the voter's end of the protocol: we have from Properties 3, 4 and 5 that no information about the state of any vote can be gained by the attacker, even after the election is complete (exempting the small probability of a tally representing a 100% vote for one candidate).

Property 8 (Vote Privacy). *Neither the authorities nor any other participant should be able to link any ballot to the voter having cast it,* unless *the protocol to revoke anonymity has been invoked* Alice casts her vote in `votetoken` over an anonymous channel. Hence, as long as no identifying data is provided with her vote, it is not possible to link Alice to her ballot.

As such, we now discuss the form of Alice's ballot. When Alice registers to vote, her TPM provides her with a pseudonym $N_{\mathbb{R}}$, with which to identify herself. Communications with $\mathbb{R}$ are in the clear, meaning Alice's machine can be identified. However, when she votes (i.e., uses the DAA *sign* protocol), her TPM generates a completely different pseudonym $N_{\mathbb{T}}$, which is then proven to represent the same values Alice gained attestation on. Any further transactions with $\mathbb{T}$ are signed with a fresh attestation identity key or another asymmetric key, generated by the TPM and unlinkable to Alice. The talliers send Alice back Designated Verifier Re-encryption Proofs, and this is the end of their interaction with her. As long as we trust the anonymous channel to be anonymous, they cannot link her to her ballot. We already have from the properties above that no other observer can link Alice to her ballot, either.

In the `vote2` version of the protocol, with revocable anonymity, whenever Alice's identity is transmitted to $\mathbb{T}$, it is encrypted with the Judge's public key as $\overline{\mathsf{id}} = \{\mathsf{id}\}_{\mathsf{Judge}}$. This encryption is done by $\mathbb{R}$, whom we trust to operate correctly. Thus, unless the protocol to revoke anonymity is invoked by the Judge, Alice's anonymity is preserved.

Property 8.1 (Revocable Anonymity). *It should be possible for an* authorised entity *(or collaboration of entities, for us) to reveal the identity of any* single *voter by linking her vote to her* We have from Property 8 that Alice's privacy is maintained at all times by the protocol, unless the Judge authorised revocation. In this case, a member of $\mathbb{T}$ contacts the Judge with appropriate evidence to justify revocation. We must of course assume that the Judge only revokes anonymity under suitable circumstances.

Property 9 (Remote Voting)**.** *Voters should not be restricted by physical location* Our protocol permits Alice to vote from any location with a computer that has a TPM, provided she has registered at some point beforehand, and can vote once unobserved. Note that our protocol also prevents an observer from knowing if Alice has voted successfully, even if the observer is physically watching Alice.

5.5 Summary

In this chapter, we have presented the second of our electronic voting protocols, detailing a novel solution which uses trusted computing to assure the security of a remote voting client. Again, we achieve the optional availability of revocable anonymity, using the TPM to ensure that the identity Alice when voting is indeed hers. We satisfy all of the standard properties required of e-voting protocols.

An interesting side effect of revocable anonymity in our protocols so far is that the traced voter has no way of *knowing* she has been traced, and can thus not hold the authorities to account in case of fraudulent deanonymisation. In the next chapter, we introduce a protocol which solves this problem.

6 Making Anonymity Auditable*

Chapter Overview

A number of fields in computer security consider the anonymity of protocol users to be of critical importance: in digital cash and electronic commerce, it is important that rogue users should not be able to trace the spender of a coin, or to link coins that user has spent with each other. In anonymous fair exchange protocols, multiple parties exchange items with one another, whilst wishing to remain anonymous (sometimes for obvious reasons). In electronic voting, the voter must remain unlinkable to their vote.

However, designers of each of these classes of protocol must consider that there are sometimes occasions when a user's anonymity must be *revoked* — a coin might be maliciously double-spent, or used for an illegal purchase; a party could renege on their promise as part of an exchange protocol; a voter may attempt to vote twice, or may not be a legitimate voter at all. The point of

*This chapter is an extended version of work presented at the Fifth International Conference on Trust and Trusted Computing (Smart and Ritter, 2012).

this chapter is not to consider for what reason anonymity revocation is required, though: instead, we suggest that, generally speaking, users whose anonymities are revoked should be *made aware* of this fact. In this chapter, we present a solution to this problem, which is essentially a digitized version of the "sealed envelope problem" discussed in Ables and Ryan (2010).

Let us consider the physical, paper abstraction of the problem. Alice lives in a country where it must be possible to link her identity to her vote (though only authorised entities should be able to make this distinction). When she collects her ballot paper, her identity is sealed inside a tamper-evident envelope, and the serial number of her ballot paper is written on the outside. The envelope is stored securely. Alice votes. Some time later, for whatever reason, someone may wish to trace Alice's ballot back to her. After the election, Alice may wish to see whether her anonymity has been revoked or not. To do this, she merely requests to see the appropriate envelope from the authorities (i.e., that with her ballot serial number on it), and verifies that the envelope is still sealed.

We can apply this abstraction to a number of other fields, and it particularly makes sense when considering payment for goods (we discuss this more in Section 6.4). However, digitising the (auditable) sealed envelope is not at all trivial: it is intuitively not possible to simply give the authorities an encrypted copy of Alice's identity: if the key is provided with the ciphertext, then Alice has no way to know whether it has been used. If the key is *not* provided, then the authorities cannot do anything with the ciphertext anyway, without contacting Alice (who, as a rogue user, may deliberately fail to provide information) (Ables and Ryan, 2010). As a result, we must consider that some sort of trusted platform is required, in order for Alice to be convinced that her anonymity has not been revoked. In this chapter, we detail a protocol which uses *trusted computing*—specifically, the TPM—to assure Alice in this way.

Related Work

The work presented in this chapter is potentially relevant to a wide range of fields where revocable anonymity is important: digital cash, fair exchange, and electronic voting. We do not specifically address *any* of these areas in this chapter, as the way in which they use the identity of the user is

unimportant to us: it is the similarity in the need for the user's anonymity that matters. Very little existing work considers auditable revocable anonymity: as we discussed in Chapter 2, Kügler and Vogt (2003) describe an electronic payment protocol in which the spender of a coin can determine (within a fixed period) whether their anonymity is revoked or not. Although the protocol is attractive, it requires knowledge *a priori* of who is to be traced—something which is not possible in fields such as electronic voting. More generally, Moran and Naor (2010) discuss many high-level theoretical implementations of cryptographic "tamper-evident seals", but do not go into detail as to how these would be realised (and seemingly place a lot of trust in the entity responsible for generating seals).

Ables and Ryan (2010) discuss several implementations of a "digital envelope" for the storage of escrowed data using the TPM. Their second solution is appealing, and uses a third party with monotonic counters. However, their solution allows only a single envelope at a time to be stored (as the TPM only permits the usage of one monotonic counter at a time), and also would require Alice herself to generate her identity (something which would not be appropriate for us).

The work of Sarmenta et al. (2006) on virtual monotonic counters using a TPM is crucial to our work, as we use a new virtual monotonic counter for each anonymous user, allowing each to track their own anonymity. We discussed virtual monotonic counters more in Section 3.2.3.

Motivation and Contribution

In this chapter, we introduce a new protocol, not tied to any specific class of user-anonymous security protocols (electronic commerce, voting, et cetera), which uses the TPM to assure a user of whether or not their identity has been revealed: a property we name *non-repudiation of anonymity revocation*. Our motivation is clear: if we are to have protocols providing anonymity revocation, then it must be possible for a user to determine when their anonymity is revoked. The reasoning for this is twofold: not only does a user have the right to know when they have been identified (generally, as a suspect in a crime), but the fact that anonymity revocation is traceable is also beneficial:

> …the detectability of inappropriate actions and accountability for origination suffices to prevent misbehaviour from happening (Weber and Mühlhäuser, 2011, p. 5)

Though protocols exist in electronic commerce which permit this (Kügler and Vogt, 2003, for example), the techniques used are not widely applicable, for reasons discussed above. We consider preliminary discussions of "escrowed data" stored in a digital envelope which use *monotonic counters* (Ables and Ryan, 2010), and discuss the use of *virtual monotonic counters* (Sarmenta et al., 2006) to allow multiple tokens to be securely stored by a single entity.

6.1 Chapter Structure

Having already discussed trusted computing and the TPM in Section 2.4, and *monotonic counters* in Section 3.2.3, in Section 6.2, we discuss our trust requirements for the protocol, which itself is presented in Section 6.3. We address the applicability of the protocol to different computer security discourses in Section 6.4, and give a short discussion on the security of the protocol in Section 6.5. Finally we conclude.

6.2 Trust Model

In this chapter, we make the following trust assumptions:

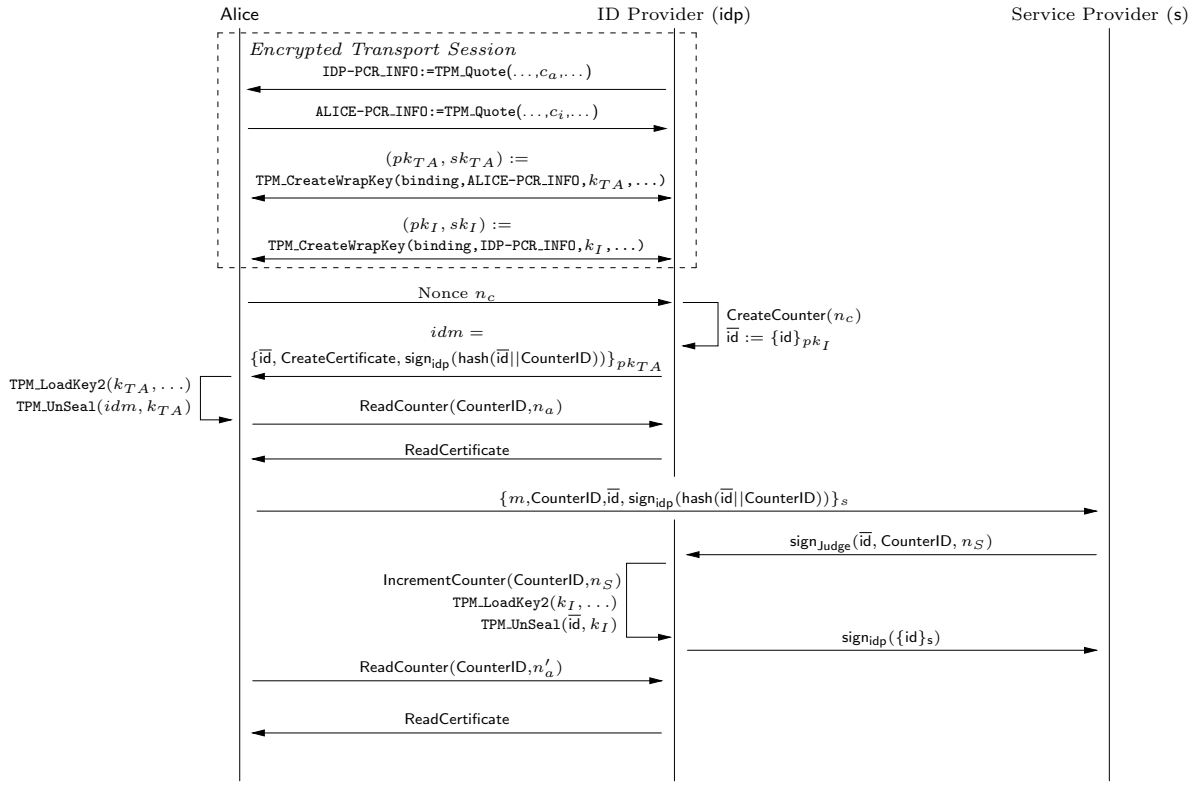
1. Alice and the identity provider idp—defined below—trust the TPM in Alice's machine, by virtue of it attesting to its state (and therefore, the state of Alice's machine)

2. All users trust idp, by virtue of it attesting to its state (and therefore, the state of idp's machine)

3. The judge is trusted to only authorise anonymity revocation where necessary

In a strict sense, it is not necessary for users to deliberately place trust in any TPM (whether it is in the identity provider's machine, or the user's): both the user's and the identity provider's TPMs have the ability to verify the correctness of the other's TPM and host machine, where the

Figure 6.1 Our Revocation Audit Protocol.



TPM itself is assumed to be a tamper-resistant hardware module. Instead, therefore, any trust we place must be in *the manufacturer of* the TPM, to construct such a device according to its correct specification.

6.3 Protocol

We begin by explaining our protocol from a high level, and then go into more implementation specific detail. Note that we assume the availability of standard public key cryptographic techniques, hashing and signature protocols. Our scenario is as follows. Alice wishes to engage in a user-anonymous protocol with a *service provider*, s: Alice normally remains anonymous, but s has some interest in revoking her anonymity under certain circumstances (s can obtain a signed request for the user's identity from a judge). Alice would like to know whether or not her anonymity has been revoked at some point after her interaction with s is complete.

In order to present a solution, we introduce a third party, the *identity provider*, idp. The

identity provider runs trusted hardware, and attests to the state of his machine in an authenticated encrypted transport session with Alice's TPM. Once Alice is assured that she can trust idp's machine, and idp is likewise assured of the trustworthiness of Alice's machine, idp generates a virtual monotonic counter specifically for Alice's identity, using a nonce sent by Alice. He then encrypts Alice's identity using a key generated by Alice's TPM. This is concatenated with a certificate produced by the creation of the counter, hashed, and signed. The signature, certificate and encrypted ID—which we will refer to as a *pseudonym*—are sent to Alice, encrypted with a binding wrap public key to which only her TPM has the private counterpart.

Alice now reads the counter generated for her. She can then send whatever message is necessary to s, along with the particulars of the counter relating to her ID, and idp's signature thereof. The service provider is able to verify the validity of the signed hash on Alice's identity, and can store it for further use.

Should s request to view Alice's identity, he contacts idp with a signature generated by a judge, on the pseudonym and particulars of the certificate (the details originally sent to him). The protocol dictates that idp first increments the virtual monotonic counter associated with the certificate received, and can then load the appropriate key, and decrypt Alice's identity. Alice is later able to request the value of her monotonic counter once again, allowing her to determine whether or not her anonymity was revoked.

6.3.1 Implementation Steps

We now present a more detailed implementation. A diagram for the protocol is give in Figure 4.2. The protocol can be split into two stages: in the first, Alice registers her identity with idp, and receives a pointer to a virtual monotonic counter back. In the second, she interacts with s, who may wish to obtain her identity. She is then able to audit this process.

6.3.1.1 Stage 1

Alice begins with her TPM and the TPM of the identity provider, idp, engaging in an *encrypted transport session*[1]. She invents a nonce, c_a, and challenges idp's TPM to reveal the state of a number of its *platform configuration registers* (PCRs—a set of protected memory registers inside the TPM, which contain cryptographic hashes of measurements based on the current state of the host system; see Section 2.4), using the `TPM_Quote` command (with c_a being used for freshness). Alice can use this information to determine if the TPM is in a suitable state (i.e., if its host machine is running the correct software). The identity provider's TPM does the same with Alice's TPM, using a different nonce c_i. In this manner, both platforms are assured of the trustworthiness of the other.

Alice proceeds to have idp's TPM generate a fresh binding RSA keypair $k_I = (pk_I, sk_I)$ using the `TPM_CreateWrapKey` command, binding the key to the PCR information she acquired. This ensures that only a TPM in the same state as when the `TPM_Quote` command was executed is able to open anything sealed with pk_I. Similarly, idp's TPM has Alice's TPM generate a binding wrap keypair $k_{TA} = (pk_{TA}, sk_{TA})$, where the private key is accessible only to Alice's TPM.

Next, idp receives a nonce n_c from Alice. He then creates a *virtual monotonic counter* (Sarmenta et al., 2006), which he 'ties' to Alice's identity, using the CreateNewCounter command with n_c. This returns a CreateCertificate, detailing the ID number of the counter, CounterID, and the nonce used to create it. idp proceeds to produce a *pseudonym* $\overline{\mathsf{id}} = \{\mathsf{id}\}_{pk_I}$ for Alice, an encryption of her identity (which we assume it knows) using the `TPM_Seal` command and the binding wrap key pk_I. $\overline{\mathsf{id}}$ and the ID of the counter, CounterID, are concatenated and hashed. The signed hash, pseudonym $\overline{\mathsf{id}}$ and the aforementioned CreateCertificate are sent to Alice, encrypted with the binding wrap key pk_{TA} generated for her TPM. The ID provider stores CounterID and $\overline{\mathsf{id}}$ locally. Alice has her TPM decrypt the message she receives, and then verifies the hash. Note that only Alice's TPM, in the correct state, can decrypt the message sent to her.

Finally, Alice generates a fresh nonce n_a, and contacts idp to request the value of the counter, via

[1] We note that idp could also undergo *direct anonymous attestation* (Brickell et al., 2004) with Alice to attest to the state of his machine. However, this is unnecessary for us, as neither Alice nor idp need to (or could) be anonymous at this stage.

the ReadCounter(CounterID, Nonce) command. She receives back a ReadCertificate containing the counter's value, the CounterID and the nonce she sent.

6.3.1.2 Stage 2

The second stage, which can happen at any time in future, is where Alice communicates with whichever service provider she chooses (note that she may choose to use the same id token with multiple service providers, or may generate a new token for each—it would obviously be sensible to do the latter, to prevent linkability between service providers). Where Alice's message (which might be a tuple containing her vote, or a coin, or some exchangeable object) is represented by m, she sends the tuple

$$\{m, \mathsf{CounterID}, \overline{\mathsf{id}}, \mathsf{sign}_{\mathsf{idp}}(\mathsf{hash}(\overline{\mathsf{id}}||\mathsf{CounterID}))\}_s$$

to s. Note that the whole message is encrypted with the public key of the service provider, preventing eavesdropping. The message m is further processed (how is outside of the scope of this chapter). The signed hash is examined to confirm that it is indeed a valid signature, by idp, on the pseudonym and Counter ID provided. The service provider can then store $\langle \mathsf{CounterID}, \overline{\mathsf{id}} \rangle$ for later use.

Now, Alice can, at any point, check the value of her virtual monotonic counter. The service provider may wish to discover her identity, and so will seek a signed request from a judge, generating a nonce n_S. He sends this request, $\mathsf{sign}_{\mathsf{Judge}}(\overline{\mathsf{id}}, n_S, \mathsf{CounterID})$, to idp. Note that in order to decrypt Alice's pseudonym, idp must use the key k_I—bound to the correct state of his TPM's PCRs—which Alice selected. This means that he needs to be in the correct state. He begins by incrementing Alice's virtual monotonic counter using the command IncrementCounter(CounterID, n_S), and then loads the appropriate key k_I using the `TPM_LoadKey2` command. He can then decrypt Alice's identity using `TPM_UnBind`. Finally, idp returns id, encrypted for s. Again, what s does with Alice's identity is outside of the scope of this chapter.

At any later time, Alice can check the virtual monotonic counter value, by contacting idp and

executing the ReadCounter command with a fresh nonce n'_a. If idp was correctly following the protocol (which, using a verified TPM, he must have been), Alice will know if her identity has been revealed by whether the value of the counter has increased.

A key point of the protocol is that the identity provider is automatically trusted to follow it, as a consequence of the encrypted transport session in Stage 1. When Alice quotes the PCRs of the identity provider's TPM, she makes it generate a key bound to the correct machine state that it is currently in (presumably, Alice would terminate any session where an erroneous result of `TPM_Quote` was reported). Even if idp were to become corrupted after the encrypted transport session, this corruption would alter its TPM's PCRs, protecting Alice's identity from rogue decryption.

6.4 Applicability

In this section, we discuss some use cases for the protocol: as mentioned earlier, we believe it to have a number of areas of applicability. Here we focus on digital cash and electronic voting, two classes of protocol where anonymity is critical.

6.4.1 When Does **Alice** Request a Pseudonym?

We mentioned in Section 6.3.1.2 that Alice is free to have idp generate an unlimited number of pseudonyms for her, or just one, depending on her preference. Common sense dictates that, should Alice wish the services she interacts with to be unable to link her transactions together, she should generate a fresh pseudonym for each service she uses. For services which a user uses only once (say, participating in an election), this solution is sufficient. For those which she uses multiple times—such as spending multiple coins in a digital cash system—we consider whether a solution requiring Alice to contact idp multiple times for different pseudonyms is suitable. Digital cash protocols such as (Jakobsson and Yung, 1996) typically secure a spender's identity by encrypting it with a key to which only one, trusted, entity has access. When coins are withdrawn, the identities of those coins are stored with the encrypted ID of their owners in a database. Consequently, as

in Jakobsson and Yung (1996), though the digital coin itself does not contain Alice's identity, it contains pointers which which her identity can be looked up in the database.

We note that, in Jakobsson and Yung (1996), whenever Alice withdraws a coin, she encrypts her identity using fresh symmetric keys for two separate parties: the Bank and the Ombudsman, both of whom have to cooperate to later retrieve her anonymity. In fact, our protocol fits very well into this model. Alice still selects two fresh symmetric keys, but now encrypts not her plaintext ID, but the tuple

$$\langle \mathsf{CounterID}, \overline{\mathsf{id}}, \mathsf{sign}_{\mathsf{idp}}(\mathsf{hash}(\overline{\mathsf{id}}||\mathsf{CounterID}))\rangle,$$

obtained from idp. As idp is trusted to legitimately produce signatures on identities, the Bank and Ombudsman can trust the encrypted ID to be legitimate, and issue the coin as before. Should revocation be required, the Bank now simply contacts idp, allowing Alice to determine that this has occurred.

The advantage here is that Alice's withdrawn coins remain unlinkable—her ID is not encoded into them, and every instance of her ID stored by the Bank is not only encrypted with the key idp generated for it, but also with session keys generated by Alice. We note, of course, that Jakobsson and Yung (1996) is now quite dated. However, it represents a class of digital cash protocol in which the spender's identity is stored encrypted in a database, and is used here for its simplicity. A range of other digital cash systems could use our protocol in the same way (Camenisch et al., 1997; Chen et al., 2011; Tan, 2011; Wang and Lu, 2008), or by simply storing the pseudonym in the coin (Fan and Liang, 2008; Fuchsbauer et al., 2009; Hou and Tan, 2005; Pointcheval, 2000).

6.4.2 Digital Cash Examples

If we take any digital cash protocol where the identity of the coin spender is in some way encrypted (whether stored on a remote server (Jakobsson and Yung, 1996) or encoded into the coin itself (Pointcheval, 2000)), we can envisage a situation in which a user either spends a digital coin twice, or participates in an illegal transaction. An authority will have some interest in this, and thus requests that the Bank trace the coins spent by the user, in order to identify her.

In the case of the protocols listed above, the identity of the user is simply decrypted (albeit by two separate authorities in the first case). The user has no way to know that she was traced, until she is apprehended! Now, we modify each protocol such that:

- in the case of protocols where the spender ID is encoded onto the coin, the coins instead contain the user's identity—encrypted using the wrap key made for idp—and the CounterID, with the signed hash of both;

- in the case of a database storing the spender ID, with a lookup value in each key, we proceed as discussed above, with the spender providing the idp-encrypted ID token which is then stored in the database.

This done, the coin spender knows that each coin can only be linked back to her with the cooperation of idp, who (since he is following the protocol) must increment the appropriate counter, allowing the spender to know if she is identified. Note that a protocol providing revocation auditability already exists (Kügler and Vogt, 2003), but requires knowledge *a priori* of who is to be traced, making the protocol unsuitable for other applications.

6.4.3 Electronic Voting Example

Our work on revocable anonymity in electronic voting, described in Chapters 4 and 5, stores the voter's identity in an encrypted manner in the ballot. If instead we store the encrypted ID, with the CounterID and signed hash of both, we achieve the same property as above: if the authorities need to trace a voter, they contact the identity provider. If a voter is traced, they know that they will be able to determine this was the case, because the identity provider will have incremented their virtual monotonic counter.

An interesting problem is how to deal with coercion resistance: if Alice receives an encrypted identity from idp, and then sends it to a vote tallier who places it on the bulletin board unchanged, then a coercer can see that Alice has voted (this is undesirable if we wish to prevent forced-abstention attacks). In protocol `vote2`, permitting revocable anonymity, revocation is effected by

having Alice send the tuple $\langle \overline{\mathsf{id}} = \{\mathsf{id}\}_{\mathsf{Judge}}, \mathsf{Sign}_{\mathbb{R}}(\overline{\mathsf{id}})\rangle$ to the talliers. The ciphertext $\overline{\mathsf{id}}$ is produced by the registrar, $\mathbb{R}$, during registration.

This is followed by an encrypted transport session between the voter's TPM and a Tallier, in which a sealing wrap key used to encrypt designated verifier proofs of re-encryption is produced. Our change to the protocol is again quite small. In the registration phase, once the "join" stage of the protocol is complete, Alice sends her idp-encrypted $\overline{\mathsf{id}}$ to $\mathbb{R}$, who performs an ElGamal encryption of it using the Judge's public key. Before the talliers post this ciphertext to the bulletin board, it is randomly re-encrypted. Should revocation be required, the co-operation of both the Judge and idp is required, and Alice will again be able to see that this has occurred.

6.5 Analysis

In this section we briefly discuss the security properties of the protocol. The main property that we achieve is that Alice is always able to determine whether her anonymity is revoked or not (non-repudiation of anonymity revocation). This property is satisfied as a result of the knowledge that, having attested to the state of his TPM (and hence, the software being run on the host), idp will either:

- act according to the protocol specification, or
- be unable to decrypt Alice's identity.

Our reasoning is as follows. If the Identity Provider adheres to the specification, he generates a counter for Alice's identity using a nonce she supplies. He encrypts her identity using a keypair which can only be used again by a TPM in the same state which Alice originally accepted.

The information that idp generates to send to Alice must be correct, otherwise idp is deviating from the protocol. It follows that, when s requests Alice's anonymity to be revoked, idp must first increment the associated counter. If idp *does* deviate from the protocol, he will not be able to use the same key k_I later on to decrypt Alice's identity, as that key is bound to his original TPM state (which would change if different, or malicious, software were used).

Thus, the most a rogue idp could achieve is suggesting Alice's anonymity has been revoked when it has not (i.e., tampering with the counter), opening up idp to further questioning (it is hence not in the identity provider's interest to lie to Alice in this way). Since the counter must always be incremented before Alice's identity is decrypted, Alice will always know when she has been identified, by querying the counter.

We next consider Alice's interaction with s. In her communication with s, Alice provides her pseudonym and the counter ID tied to it, together with a signed hash of these values (as originally provided to her by idp). This convinces s that the identity provided is genuine. This leads us to the issue of eavesdropping attacks, allowing a user to illegitimately obtain the pseudonym of another user, and thus 'frame' an innocent victim for a crime. Note that without identifying Alice immediately, s cannot be further convinced that the pseudonym is indeed *hers*. However, our protocol prevents this problem from arising: in the message *idm* sent from idp to Alice, Alice's pseudonym and counter information are encrypted using a binding wrap key, meaning that only her TPM can obtain these values. The only other message where these two values are together is in Alice's communication with s, and here, the entire message is encrypted for s.

The message containing Alice's actual identity is signed by idp before being sent back to s. Hence, providing s trusts idp, he will always obtain Alice's legitimate identity by following the protocol. We might consider that s does *not* trust idp, in which case we could request that s and idp also undergo some sort of attestation, like that between Alice and idp. In the case of the digital cash example presented earlier, we could require that the Bank and Ombudsman each force idp to attest to its state.

6.5.1 Trustworthiness of the Service Provider

Note that, as we have already mentioned, we do not consider how s behaves, as it is outside of the scope of this protocol. However, we now discuss a possible course of action to prevent a rogue s replaying the counter and pseudonym values sent to him by an honest user. In order to mitigate this issue, we need to force the pseudonym's actual owner to prove her ownership. We therefore

alter some of the messages in the protocol (numbered according to Figure 4.2, where messages 10a–d come between messages 10 and 11):

7. idp→Alice: $\{\overline{\text{id}}, \text{CreateCertificate}, \text{sign}_{\text{idp}}(\text{hash}(\overline{\text{id}} \parallel \text{hash}(\text{CounterID})))\}_{pk_{TA}}$

8. Alice→idp: $\{\text{ReadCounter}(\text{CounterID}, n_a)\}_{pk_I}$

9. idp→Alice: $\{\text{ReadCertificate}\}_{pk_{TA}}$

10. Alice→s: $\{m, \overline{\text{id}}, \text{hash}(\text{CounterID}), \text{sign}_{\text{idp}}(\overline{\text{id}} \parallel \text{hash}(\text{CounterID}))\}_s$

10a. s→Alice: c_{ctr}

10b. Alice→s: $\text{hash}(\text{CounterID} \parallel c_{ctr})$

10c. s→idp: $\overline{\text{id}}, c_{ctr}$

10d. idp→s: $\text{hash}(\text{CounterID} \parallel c_{ctr})$

11. s→idp: $\text{sign}_{\text{Judge}}(\overline{\text{id}}, n_S)$

These changes are appropriate if we wish to prevent a rogue s from gaining an $\langle\overline{\text{id}}, \text{CounterID}\rangle$ pair with which to frame another user. We begin by altering what idp sends to Alice, such that the signed hash now itself contains a hash of CounterID. Both the request and result of reading the counter are encrypted for idp's and Alice's TPM respectively.

The messages from 10 onwards are the most important. Rather than sending her counter's ID in the clear for s, Alice sends a hash of it, which fits in with the signed hash provided by idp. s now returns a challenge c_{ctr}, which Alice hashes with CounterID and returns. In 10c and 10d, s sends the pair $\langle\overline{\text{id}}, c_{ctr}\rangle$ to idp, who looks up $\overline{\text{id}}$ and returns a hash of its associated CounterID concatenated with the challenge. This allows s to ensure that Alice really is the owner of the pseudonym and counter ID she provided. No further changes are necessary, as this prevents s from stealing Alice's pseudonym and counter ID: s would be unable to generate message 10b as he never sees CounterID in the clear. Note that consequently, message 11 also needs to change.

In this section, we have discussed the security properties of our work. Note that changes to mitigate against a corrupt service provider are only appropriate where untrustworthy service providers are a risk—hence we do not include these changes in the main protocol.

6.6 Conclusions and Future Work

In this chapter, we have presented work on a protocol which allows users of a protocol providing revocable anonymity to audit whether or not their anonymity is revoked. We have shown how virtual monotonic counters can be used on an authenticated host to track anonymity revocation, for use with any other class of security protocol requiring revocable anonymity. Further, we addressed how to mitigate the actions of a corrupt service provider. This work makes significant steps in auditable anonymity revocation, a field which has not been considered in detail before.

There are factors which we would like to consider in future work. Some of those are motivated by the issues Sarmenta *et al.* discuss regarding log-based virtual monotonic counters in Sarmenta et al. (2006). The counters are non-deterministic, being based on the single counter in use by the TPM in any one power cycle. This means that counter increment values are unpredictable—not a problem for our application, but potentially a cause of high overhead. Indeed, the ReadCertificate for a counter would include "the log of *all* increments of *all* counters…since the last increment". The size of such a certificate could be substantial. Power failures mid-cycle on idp could also cause the counters to become untrustworthy.

These issues are mitigated by the idea of *Merkle hash tree-based* counters (Sarmenta et al., 2006, pp. 34–6) which would require changes to the TPM's API. It is for this reason that we did not adopt this solution, but would instead look to it for future work. We would also like to consider a formal analysis of the security properties of the protocol.

We feel the protocol we have presented has wide-ranging applicability to a number of user-anonymous protocols—particularly those in digital cash and electronic voting—allowing all users subject to revocable anonymity to be assured of whether or not they can be identified. In the next chapter, we formalise a number of the requirements set out in Chapter 2 using the applied

pi calculus and a modelling tool based upon it, ProVerif, in order to prove the security of one of our protocols.

7 Formalisation of Security Properties

Chapter Overview

In this chapter, we introduce the manner in which we prove the security properties of one of our protocols. We model our protocol using the automated reasoning tool *ProVerif* (Blanchet, 2001; Blanchet et al., 2005), a tool capable of proving reachability properties, correspondence assertions and observational equivalences.

ProVerif takes as input protocol models defined in the *applied pi calculus* (Abadi and Fournet, 2001), a language used to model concurrent systems, built explicitly for modelling cryptographic protocols. We begin this chapter with a summary of our justifications for selecting ProVerif to formalise our work. In Section 7.3, we discuss the applied pi calculus, and then move onto a thorough discussion of ProVerif in 7.4. We then discuss how we have used ProVerif to formalise the work discussed in Chapter 4, and prove a number of important security properties.

7.1 Why Formal Modelling?

In this chapter, we explore our use of the *applied pi calculus* to formalise our requirements, and prove the security of our work. Our decision to use formal modelling stems from the belief that automated proofs more effectively guarantee that security properties are satisfied, without the time-consuming process of interactive theorem-proving or alternative mathematical proofs (which frequently lead to human error, introducing flaws). Using a standard mathematical proof, it is often difficult to prove a requirement in all possible situations, and is even more difficult to 'prove' that proof to an observer. Consequently, in our work we write a formal model for our protocol, and then perform various tests upon it. This leaves us with two problems:

1. Writing a model that is close enough to the protocol being verified that important features of it will not be missed, and

2. Formalising certain requirements is sometimes difficult, if not impossible. This is a separate problem from the first: we begin by writing a model that is sufficiently close to the protocol to capture its properties, disregarding any requirements we have. Only then do we consider the specification of the properties we are trying to prove, which may lead to a reformulation of the model.

7.2 Why Applied Pi?

In this chapter, we use the applied pi calculus, and automated reasoning tool ProVerif, to show that several properties are satisfied by the protocol in Chapter 4. We considered other languages: CSP (Hoare, 1978) and CCS (Milner, 1989) to name two.

The π-calculus (Milner et al., 1992), an ancestor of the applied-pi calculus, added the ability to describe concurrent, changeable processes. Though powerful, the π-calculus alone was not suitable for the properties being proved in our work, not least because it only permits the transmission of channel names between processes. We might have chosen the spi-calculus (Abadi and Gordon, 1999), which extended the π-calculus by providing a formal notation for reasoning

about cryptographic protocols, giving primitives for simple cryptographic processes. The limited set of cryptographic primitives provided by the spi-calculus (for symmetric and asymmetric encryption) meant that extensive work was required for each inclusion of new cryptographic operations, however. Soon after, the spi-calculus was further extended into the applied-pi calculus, which parameterised the spi-calculus using a signature and equation system for data structures, allowing value passing and providing an equational theory over terms and functions. The calculus allows for the modelling of observational equivalence, reachability and correspondence-style properties.

Our choice of calculus partially stems from the considerable amount of existing work in formal analysis of electronic voting systems using the applied pi calculus (Kremer and Ryan, 2005; Delaune et al., 2006; Backes et al., 2008; Kremer et al., 2010). This has allowed us to draw upon existing proof techniques. The calculus also has a number of automated reasoning tools based upon it, including our choice, ProVerif, which allows not only for the proof of standard reachability properties, but also (sometimes) for the proof of biprocess observational equivalence.

7.3 The Applied Pi Calculus

We begin with a discussion of the applied pi calculus, on which it is possible to base ProVerif's input. The calculus is built upon the earlier *pi calculus*, adding functions and equations, and being designed explicitly for the study of security protocols. Note that this section is adapted from the work of Abadi and Fournet (2001).

7.3.1 Syntax and Semantics

The calculus defines a *signature* Σ as a finite set of function symbols, each with an arity (where a function symbol with zero arity is a constant). Given the signature, and infinite set of names and variables, terms are defined by the grammar

$L, M, N, T, U, V ::=$	Terms
$a, b, c, \ldots, k, \ldots, m, n, \ldots, s$	Name
x, y, z	Variable
$f(M_1, \ldots, M_l)$	Function application

The symbol f represents the functions of Σ, and l is hence the arity of f.

A term is denoted as *ground* when it does not have free variables, and tuples $u_1, \ldots, u_l$ and $M_1, \ldots, M_l$ are abbreviated to $\tilde{u}, \tilde{M}$, accordingly.

The grammar for processes is given next, where c is a channel name:

$P, Q, R ::=$	Plain Processes
$\mathbf{0}$	Null process
$P \mid Q$	Parallel Composition
$!P$	Replication
$\nu n.P$	Name restriction
if $M = N$ *then* P *else* Q	Conditional
$c(x).P$	Message input on channel c
$\bar{c}\langle N \rangle.P$	Message output on channel c

The process $\mathbf{0}$ does nothing; $P|Q$ represents process P running simultaneously with Q; $!P$ behaves as an infinite number of Ps running in parallel (cf. $P|!P$). The *new* construct $\nu n.P$ makes a new, private name n, and then behaves as P. Also of note are $c(x).P$ and $\bar{c}\langle N \rangle.P$, which represent the input and output of a message x and a term N, respectively, on channel c, followed by the execution of P. This leads to a further extension of the calculus to allow for *active substitutions*:

$A, B, C ::=$	Extended Processes
P	Plain process
$A \mid B$	Parallel Composition (as above)
$\nu n.A$	Name restriction
$\nu x.A$	Variable restriction
$\{^M/_x\}$	Active substitution

$\{^M/_x\}$ represents the substitution replacing the variable x with term M, thus meaning that the process $\nu x.(\{^M/_x\}|P)$ is exactly equivalent to *let* $x = M$ *in* P.

The authors note that, as usual, names and variables have scope, delimited by restrictions and inputs. $fv(A), bv(A)$ represent the free (resp. bound) variables of extended process A, and $fn(A), bn(A)$ represent free (resp. bound) names. A process is *closed* when every variable is either bound *or* defined by an active substitution.

The authors define a *frame* as an extended process built up from $\mathbf{0}$ and active substitutions $\{^M/_x\}$ by parallel composition and restriction. The *domain* of a frame, $dom(\varphi)$ (where φ, ψ represent frames), is the set of variables exported by that frame (variables x for which the frame contains a substitution not under restriction on x). In order to map an extended process A to a frame $\psi(A)$, we replace every plain process in A with $\mathbf{0}$, giving an approximation of the static information exposed to the attacker.

7.3.2 Operational Semantics

We discussed above the manner in which a signature Σ is constructed. Abadi and Fournet progress to give that signature an equational theory, i.e. an equivalence relation on terms, closed under substitutions of terms for variables. A *context* is defined as an expression with a hole, and an *evaluation* context as a context whose whole is not under replication, a conditional expression, an input or an output. A context $C[\_]$ closes A when $C[A]$ is closed.

We begin with *structural equivalence* ($\equiv$), the smallest equivalence relation on extended processes, such that

$$
\begin{array}{lrcl}
\textsc{Par-0} & A & \equiv & A \mid 0 \\
\textsc{Par-A} & A \mid (B \mid C) & \equiv & (A \mid B) \mid C \\
\textsc{Par-C} & A \mid B & \equiv & B \mid A \\
\textsc{Repl} & !P & \equiv & P \mid !P \\
\textsc{New-0} & \nu n.0 & \equiv & 0 \\
\textsc{New-C} & \nu u.\nu v.A & \equiv & \nu v.\nu u.A \\
\textsc{New-Par} & A \mid \nu u.B & \equiv & \nu u.(A \mid B) \text{ when } u \notin fv(A) \cup fn(A) \\
\textsc{Alias} & \nu x.\{^M/_x\} & \equiv & 0 \\
\textsc{Subst} & \{^M/_x\} \mid A & \equiv & \{^M/_x\} \mid A\{^M/_x\} \\
\textsc{Rewrite} & \{^M/_x\} & \equiv & \{^N/_x\} \text{ when } \Sigma \vdash M = N
\end{array}
$$

The authors further note that structural equivalence allows every closed extended process A to be rewritten as a substitution and a closed plain process with restricted names: $A \equiv \nu \tilde{n}.\{^{\tilde{M}}/_{\tilde{x}}\} \mid P$, where $P, \tilde{M}$ have no free variables, and $\tilde{n}$ are part of the free names of $\tilde{M}$.

Next, *internal reduction* ($\rightarrow$) is the smallest relation on extended processes closed by structural equivalence such that

$$
\begin{array}{lrcl}
\textsc{Comm} & \overline{a}\langle x \rangle.P \mid a(x).Q & \rightarrow & P \mid Q \\
\textsc{Then} & \textit{if } M = M \textit{ then } P \textit{ else } Q & \rightarrow & P \\
\textsc{Else} & \textit{if } M = N \textit{ then } P \textit{ else } Q & \rightarrow & Q \text{ for any ground } M, N \\
 & & & \text{such that } \Sigma \nvdash M = N
\end{array}
$$

7.3.3 Examples

In this section, we detail how the calculus might be used to model a number of different primitives. We begin with the binary function symbol pair, for example, which represents a pair of values

(cf. $\mathsf{pair}(M, N)$), abbreviated to (M, N), and leading to unary symbols

$$\begin{aligned} \mathsf{fst}((x, y)) &= x \\ \mathsf{snd}((x, y)) &= y \end{aligned}$$

We might also represent a one-way hash function, h, where the fact that $h(M) = h(N)$ implies that $\Sigma \vdash M = N$, since the function is collision free. Other primitives include symmetric encryption, viz.

$$\mathsf{dec}(\mathsf{enc}(x, y), y) = x$$

given a symmetric key y and message x, and asymmetric, probabilistic encryption, requiring the introduction of two new unary primitives, to obtain the public and private counterparts of a key:

$$\mathsf{pdec}(\mathsf{penc}(x, \mathsf{pk}(y), z), \mathsf{sk}(y)) = x$$

given a message x, key y and random seed z (we note that omitting the z leads to deterministic asymmetric encryption). Abadi and Fournet introduce an example of asymmetric encryption:

$$\nu s.(\overline{a}\langle \mathsf{pk}(s)\rangle \mid b(x).\overline{c}\langle \mathsf{dec}(x, \mathsf{sk}(s))\rangle)$$

We can read this as a fresh seed s being generated, and the public key from it being broadcast on (the public channel) a. The second part of the process reads a value x on channel b, and outputs its decryption using the secret key part of s. We finally briefly mention the binary function symbol sign, ternary symbol check and constant ok:

$$\mathsf{check}(x, \mathsf{sign}(x, \mathsf{sk}(y)), \mathsf{pk}(y)) = \mathsf{ok}$$

Of course, one is free to add function symbols to the language, as appropriate.

7.3.4 Proof Techniques

The calculus enables a number of techniques to prove the *equivalence* (specifically, observational equivalence) of two processes. Two processes are *observationally equivalent* when they cannot be distinguished by any context. Thus, if we envisage the context as the attacker's view, we may state that various security properties can be claimed as a result of the observational equivalence of two processes. We begin with the authors' definition of observational equivalence. Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ between closed extended processes with the same domain, such that $A \mathcal{R} B$ implies

1. if A can emit a message on channel a (written $A \Downarrow a$), then $B \Downarrow a$
2. if $A \rightarrow^* A'$ then $B \rightarrow^* B'$ and $A' \mathcal{R} B'$
3. $C[A] \mathcal{R} C[B]$ for all closing evaluation contexts $C[\_]$

(Abadi and Fournet, 2001, p. 108)

Static equivalence ($\approx_s$) is the notion that two *substitutions* may be seen as equivalent when they behave equivalently when applied to terms. Static equivalence can be seen as representing the initial knowledge of the attacker.

In order to discuss processes which interact with their environment (cf. via input and output), the authors also introduce a labeled operational semantics:

$$\textsc{In} \qquad a(x).P \xrightarrow{a(M)} P\{^{M}/_{x}\}$$

$$\textsc{Out-Atom} \qquad \overline{a}\langle u \rangle.P \xrightarrow{\overline{a}\langle u \rangle} P$$

$$\textsc{Open-Atom} \qquad \frac{A \xrightarrow{\overline{a}\langle u \rangle} A' \quad u \neq a}{\nu u.A \xrightarrow{\nu u.\overline{a}\langle u \rangle} A'}$$

$$\textsc{Scope} \qquad \frac{A \xrightarrow{\alpha} A' \quad u \text{ not in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$$

$$\textsc{Par} \qquad \frac{A \xrightarrow{\alpha} A' \quad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$\textsc{Struct} \qquad \frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad B' \equiv A'}{A \xrightarrow{\alpha} A'}$$

These semantics allow us to construct fairly complex models, such as the one in Figure 7.1, which represents a simple encryption and decryption. In the example, knowledge of k released to the attacker by the operation $\overline{a}\langle k\rangle$ is sufficient to compute M from its encryption.

Figure 7.1 A simple example of labelled transitions in the Applied Pi Calculus (Abadi and Fournet, 2001, p. 109)

$$
\begin{array}{lcl}
\nu k.\overline{a}\langle \mathsf{enc}(M,k)\rangle.\overline{a}\langle k\rangle.a(z).\mathit{if}\ z = M\ \mathit{then}\ \overline{c}\langle \mathsf{oops!}\rangle & \xrightarrow{\nu x.\overline{a}\langle x\rangle} & \nu k.(\{^{\mathsf{enc}(M,k)}/_{x}\} \mid \overline{a}\langle k\rangle.a(z).\mathit{if}\ z = M\ \mathit{then}\ \overline{c}\langle \mathsf{oops!}\rangle) \\
 & \xrightarrow{\nu y.\overline{a}\langle y\rangle} & \nu k.(\{^{\mathsf{enc}(M,k)}/_{x}\} \mid \{^{k}/_{y}\} \mid a(z).\mathit{if}\ z = M\ \mathit{then}\ \overline{c}\langle \mathsf{oops!}\rangle) \\
 & \xrightarrow{a(\mathsf{dec}(x,y))} & \nu k.(\{^{\mathsf{enc}(M,k)}/_{x}\} \mid \{^{k}/_{y}\} \mid \mathit{if}\ \mathsf{dec}(x,y) = M\ \mathit{then}\ \overline{c}\langle \mathsf{oops!}\rangle) \\
 & \rightarrow & \nu k.(\{^{\mathsf{enc}(M,k)}/_{x}\} \mid \{^{k}/_{y}\}) \mid \overline{c}\langle \mathsf{oops!}\rangle
\end{array}
$$

Given the labeled semantics, the authors define *labeled bisimilarity* ($\approx_l$) as the largest symmetric relation $\mathcal{R}$ on closed extended processes such that $A\,\mathcal{R}\,B$ implies

1. $A \approx_s B$
2. if $A \rightarrow A'$, then $B \rightarrow^* B'$ and $A'\,\mathcal{R}\,B'$
3. if $A \xrightarrow{\alpha} A'$, $fv(\alpha)$ is a subset of the domain of A and $bn(\alpha) \cap fn(B) = \emptyset$ then $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ and $A'\,\mathcal{R}\,B'$

The authors go on to prove that labeled bisimilarity and observational equivalence are the same ($\approx = \approx_l$): the reader is directed to Abadi and Fournet (2001) for further details. This is important because we use labeled bisimilarity *in order to prove* observational equivalence of biprocesses. Attempting to prove observational equivalence directly is challenging as one needs a quantification over all contexts. This is not the case for labeled bisimilarity, making proofs of the labeled bisimilarity of biprocesses far simpler.

We can use the calculus to consider the effect of (and role of) the intruder in any security protocol. For example, consider an election process in which voter v_i casts a certain vote a, and $v_{j\neq i}$ votes b. If that process were to be observationally equivalent to one in which v_i votes b and v_j votes a, we could reason that the attacker could not determine how any voter voted, thus demonstrating voter privacy.

A limitation of using any hand-written calculus is that it is subject to almost inevitable human error, and it is often laborious to prove any property. For this reason, we adopt the ProVerif

tool, whose input can be in the form of the (typed) applied pi calculus, to automatically verify a number of equivalences and reachability assertions.

7.4 Using Applied Pi with ProVerif

ProVerif (Blanchet, 2001) is an automated reasoning tool, capable of proving reachability properties, correspondence assertions and observational equivalences. It takes as input a model of a protocol, written in the applied pi calculus, and translates this input into horn clauses. When translating a protocol into ProVerif source, one must of course be careful to ensure that the protocol is sufficiently captured, whilst considering the limitations of the tool (for example, neither it nor the underlying calculus can be used to model random numbers, repetition, et cetera). A further difficulty is modelling the requirements which one wishes to prove, both in the calculus and in ProVerif itself.

For the remainder of this chapter, we formally prove the security of our first protocol (discussed in Chapter 4). We have adopted the work of Kremer et al. (2010) in proving our protocol to be individually and universally verifiable: these properties are modelled in terms of events and proof of reachability properties, notions that have always been present in ProVerif.

Soundness properties (viz. eligibility, uniqueness, and inalterability) are modelled by correspondence assertions in the work of Backes et al. (2008), which we adopt to prove not only those requirements, but also coercion-resistance—and thence receipt-freeness and vote privacy—which are modelled in terms of observational equivalences.

7.4.1 Proving Reachability Properties in ProVerif

A simple example of a reachability query in (typed) ProVerif is given in Figure 7.2, for a toy protocol.

In the example, we create a channel comm, readable by all observers, and a channel privatecomm, readable only by those given explicit access to it. The channel is excluded from the adversary's

Figure 7.2 Reachability in ProVerif

```
(* simple reachability example *)

free comm:channel.
free privatecomm:channel [private].

free mess:bitstring [private].
free privatemess:bitstring [private].

query attacker(mess).
query attacker(privatemess).
process
   out(comm, mess);
   out(privatecomm, privatemess)
```

knowledge. We create two free names *mess* and *privatemess*, both excluded from the adversary's knowledge initially.

In the protocol itself (everything below **process**), we output the message comm on mess (resp. privatecomm on privatemess), and then above, query whether the adversary can derive either of mess or privatemess. As expected, when ProVerif analyses the protocol, it determines that mess can be derived by the adversary, but privatemess cannot.

7.4.2 Proving Correspondence Assertions

If we modify the example given above, we can produce a fairly simple correspondence assertion, as shown in Figure 7.3:

Most of the script is unchanged. However, we now read back in the message which we output on comm. If that message is privatemess (it is not), we execute the event mess_received, followed by second_event. Otherwise, we just execute the event second_event. Above the process, we have written

$$\textbf{query event}(\text{mess\_received}()) \Longrightarrow \textbf{event}(\text{second\_event}()).$$

This query analyses whether there is any execution of mess_received which does not occur after second_event: it states that our assertion is that this is not the case: mess_received cannot be

Figure 7.3 Correspondence Assertions in ProVerif

```
(* simple correspondence assertion *)
free comm:channel.
free privatecomm:channel [private].

free mess:bitstring [private].
free privatemess:bitstring [private].

event mess_received().
event second_event().

query event(mess_received()) ⟹ event(second_event()).

process
    out(comm, mess);
    out(privatecomm, privatemess);
    in(comm, x:bitstring);
    if x = privatemess then
            event mess_received();
            event second_event()
    else
            event second_event()
```

witnessed without having witnessed second_event first. ProVerif correctly determines that this assertion is correct. In fact, mess_received is unreachable: this can be determined separately with the query **query event**(mess_received()).

7.4.3 Observational Equivalences

We may be able to reason about properties that cannot be expressed as reachability properties or correspondence assertions using observational equivalences, where two processes P and Q have the same structure, and differ only in the choice of terms. For example, in order to test whether vote privacy is satisfied in a voting protocol, we would test whether a process in which voter v_1 votes for candidate a and v_2 votes for b is observationally equivalent to one in which v_1 votes for b and v_2 for a.

The test for observational equivalence is denoted by the **choice** operator, as demonstrated in Figure 7.4.

Figure 7.4 A non-observationally equivalent biprocess

```
(* simple observational equivalence *)
free comm:channel.
free vote_a, vote_b:bitstring.
process
        out(comm, choice[vote_a, vote_b])
```

The script above models a simple biprocess in which on one side, *vote_a* is output, and on the other, *vote_b* is output. Of course, this biprocess does not satisfy observational equivalence, as the attacker can differentiate between the two names. If we (probabilistically) encrypt the vote, by introducing functions encrypt and decrypt, and a reduction rule for them, then observational equivalence is satisfied, as shown in Figure 7.5.

It should be noted that ProVerif's ability to prove observational equivalences is rather limited, in that it can only prove that a biprocess exhibits observational equivalences. This often means that certain models have to be reformulated as biprocesses in order to allow ProVerif to work correctly.

Figure 7.5 Observational equivalences in ProVerif

```
(* simple observational equivalence *)

free comm:channel.
free vote_a, vote_b:bitstring.

type secretkey.
type pubkey.
type exponent.

fun pk(secretkey):pubkey.
fun encrypt(bitstring, exponent, pubkey):bitstring.
reduc forall message:bitstring, alpha:exponent, sk:secretkey;
        decrypt(encrypt(message, alpha, pk(sk)), sk) = message.

process
        new alpha:exponent;
        new sk_tallier:secretkey;
        let pk_tallier = pk(sk_tallier) in
        let enc_va = encrypt(vote_a, alpha, pk_tallier) in
        let enc_vb = encrypt(vote_b, alpha, pk_tallier) in
        out(comm, choice[enc_va, enc_vb])
```

7.5 Proof of Security Properties in Our Work

For the remainder of this chapter, we provide ProVerif-based proofs of various security properties for our first protocol (shown in Chapter 4). In order to do this, we adopt the work of Kremer et al. (2010) and Backes et al. (2008). We split the requirements that we prove into three categories: *reachability properties* (individual and universal verifiability), *correspondence properties* (soundness, uniqueness and inalterability), and *observational equivalences* (coercion-resistance, receipt-freeness and vote-privacy).

7.5.1 Reachability Properties

In order to satisfy soundness (below), we need to show correspondences between certain events. In order to satisfy verifiability (i.e., individual and universal), we need only show that certain events are (or are not) reachable. For this, we adopt the work of Kremer et al. (2010). In their

work, the authors add *events* to the applied pi calculus. Events are modelled as outputs $\overline{\mathtt{f}}\langle M \rangle$, where $f \in \mathcal{F}$ is an "event channel", and input to this channel can only be in the form of "event variables", e, e' (Kremer et al., 2010, p. 170).

The authors specify a reachability assertion as an event $\overline{\mathtt{f}}\langle \tilde{X} \rangle$, where $\tilde{X}$ is a series of variables and constants. If an adversary can expose this event, then the process containing it satisfies reachability. Else, the process satisfies the *unreachability* assertion $\overline{\mathtt{f}}\langle \tilde{X} \rangle$.

7.5.1.1 Formalising Verifiability

The authors formalise election verifiability in the form of three boolean tests, Φ^{IV}, Φ^{UV} and Φ^{EV}. Such a boolean test is an applied pi term, with free variables, which evaluates to true or false when ground terms are substituted for the free variables. We do not consider the third test (eligibility verifiability), as it is beyond the scope of this work. However, we produce tests, again written in ProVerif, which correspond to the tests Kremer et al. use for individual and universal verifiability.

7.5.1.2 Test for Individual Verifiability

In their work, Kremer et al. define a boolean predicate test Φ^{IV} to take parameters v (a vote), $\tilde{x}$ (a voter's knowledge), y (the voter's public credential), and z (the bulletin board entry for a vote). Φ^{IV} is a successful test if it allows a voter to identify her bulletin board entry; i.e., for all votes, if the voter with a public credential D votes for candidate s then there must be an execution of the protocol producing $\tilde{M}$ such that a bulletin board entry B satisfies

$$\Phi^{IV}\{{}^{s}/_{v}, {}^{\tilde{M}}/_{\tilde{x}}, {}^{D}/_{y}, {}^{B}/_{z}\}$$

Furthermore, the bulletin board entry determines the vote:

$$\Phi^{IV}\{{}^{s}/_{v}, {}^{\tilde{M}}/_{\tilde{x}}, {}^{D}/_{y}, {}^{B}/_{z}\} \wedge \Phi^{IV}\{{}^{s'}/_{v}, {}^{\tilde{M}'}/_{\tilde{x}}, {}^{D'}/_{y}, {}^{B}/_{z}\} \Rightarrow (s = s')$$

(Kremer et al., 2010, p. 171)

Finally, in order for individual verifiability to hold, all bulletin board entries must be distinct.

7.5.1.3 Test for Universal Verifiability

The test Φ^{UV} takes two parameters, v (a vote) and z (a bulletin board entry), and is suitable if every bulletin board entry accepted by a voter (for the test Φ^{IV}) is also accepted by an observer as valid; i.e.:

$$\Phi^{IV}\{{}^{s}/_{v}, {}^{\tilde{M}}/_{\tilde{x}}, {}^{D}/_{\gamma}, {}^{B}/_{z}\} \Rightarrow \Phi^{UV}\{{}^{s}/_{v}, {}^{B}/_{z}\}$$

Further, the observer correctly counts the vote: i.e., if the test Φ^{UV} succeeds for two votes s, s' with one bulletin board entry B, then they must be votes for the same candidate:

$$\Phi^{UV}\{{}^{s}/_{v}, {}^{B}/_{z}\} \wedge \Phi^{UV}\{{}^{s'}/_{v}, {}^{B}/_{z}\} \Rightarrow (s = s')$$

(Kremer et al., 2010, p. 171)

In order to verify that a protocol is universally and individually verifiable, voters and observers perform the tests $\Phi^{IV}$and Φ^{UV}on the bulletin board, possibly also using information they have derived from voting themselves.

7.5.1.4 The Voting Process

Kremer et al. define a voting protocol in the applied pi calculus. The protocol is defined as a voter process, V, a process K modelling honest administrators (which publishes public credentials and distributes keys); a tuple of channels $\tilde{a}$ which are private, and a context A which performs setup.

A voting process is hence specified as a tuple $\langle A, V, K[\bar{c}\langle D\rangle], \tilde{a}\rangle$, where A and K are contexts such that V, A, K do not contain event channels or variables, and D models public voter credentials. v represents the value of the vote, and is not bound in A or V. The channel c is free (Kremer et al., 2010, p. 174).

The authors note that votes are generated by a process $G \widehat{=} !\nu s.((!\bar{b}\langle s\rangle)|\bar{c}\langle s\rangle)$, which selects a vote and sends it to the voter (such that several voters can also receive this vote), also outputting

it on a public channel. The process modelling a voting protocol VP is then

$$\mathsf{VP} \widehat{=} \nu b.(A[!\nu \tilde{a}.((b(v).V)|K[\bar{c}\langle D\rangle])]|G)$$

Kremer et al. (2010) go on to define an *augmented* voting process VP^+ as a voting process incorporating reachability assertions, such that

$$\mathsf{VP}^+ \widehat{=} \nu b.(A[!\nu \tilde{a}, b'.(\widehat{V}|\widehat{K})]|G|P)|Q$$

where

$$\widehat{V} = b(v).V \circ c(z).b'(y).(\overline{\texttt{pass}}\langle(\Phi^{IV}, z)\rangle|\overline{\texttt{fail}}\langle\psi\rangle)$$
$$\widehat{K} = K[\overline{b'}\langle D\rangle|\bar{c}\langle D\rangle]$$
$$P = b(v').b(v'').c(\tilde{x}').c(\tilde{x}'').c(y').c(y'').c(z').\overline{\texttt{fail}}\langle\phi' \vee \phi''\rangle$$
$$Q = \texttt{pass}(e).\texttt{pass}(e').\overline{\texttt{fail}}\langle e_1 \wedge e_1' \wedge (e_2 = e_2')\rangle$$

$$\psi = (\Phi^{IV} \wedge \neg\Phi^{UV})$$
$$\phi' = \Phi^{IV}\{{}^{v'}/_{v}, {}^{\tilde{x}'}/_{\tilde{x}}, {}^{y'}/_{y}, {}^{z'}/_{z}\} \wedge \Phi^{IV}\{{}^{v''}/_{v}, {}^{\tilde{x}''}/_{\tilde{x}}, {}^{y''}/_{y}, {}^{z'}/_{z}\} \wedge \neg(v' = v'')$$
$$\phi'' = \Phi^{UV}\{{}^{v'}/_{v}, {}^{z'}/_{z}\} \wedge \Phi^{UV}\{{}^{v''}/_{v}, {}^{z'}/_{z}\} \wedge \neg(v' = v'')$$

where Φ^{IV} is the boolean evaluation of the test Φ^{IV}; e_1, e_2 represent projections of the pair received on the `pass` channel; `fail` and `pass` are event channels, and the plain (*linear*) process $P \circ Q$ represents an execution of P, followed by Q (Kremer et al., 2010, p. 174).

Note that in our work, we do not consider the test for *eligibility* verifiability (Φ^{EV}), and hence omit mentions of it from VP^+. We leave investigation of this property in our protocol for future work. The authors explain that the augmented voting process VP^+ satisfies election verifiability if the *unreachability* assertion $\overline{\texttt{fail}}\langle\mathsf{true}\rangle$ is satisfied, and the *reachability* assertion $\overline{\texttt{pass}}\langle(\mathsf{true}, x)\rangle$

is satisfied, providing a number of variable constraints are satisfied (we refer the interested reader to the work of Kremer et al.).

7.5.1.5 Verifiability in Our Protocol

In line with the ProVerif modelling which Kremer et al. do for a protocol similar to ours (that of Juels et al. (2005)), we structure our model in terms of a number of equations and destructors to represent encryption/decryption, designated verifier signatures, designated verifier reencryption proofs, and the like. We then have a number of processes, representing the voter, $\mathbb{T}_1$, $\mathbb{T}_2$, a nondeterministic vote generator, a keypair generator and processes which ascertain verifiability and duplicate vote posting respectively. In this model, we only create two events, pass(bool) and fail(bool), and require the reachability of pass(true), and *un*reachability of fail(true). The *voter* process is shown in Figure 7.6. Of note are the points at which it executes the pass(true) event (when the reencrypted vote received from the bulletin board is the same as that which Alice was sent by $\mathbb{T}_1$, satisfying individual verifiability) and the fail(true) event (when the signature for an encrypted ballot on the bulletin board does not match the hash of its purported encrypted vote, i.e., failing universal verifiability).

Our verification of universal and individual verifiability is further carried out by the processes *verifiabilitychecker* and *duplicatechecker*, which correspond to P and Q respectively in the augmented voting process VP^+. Each process is executed according to the model code shown in Figure 7.7. That is to say that there is an unbounded number of voters, instances of $\mathbb{T}_1$ and $\mathbb{T}_2$, and key generators. There is one instance each of *verifiabilitychecker* and *duplicatechecker*. The former of these works by reading two (individually verifiable) votes from two voter process executions, and one bulletin board entry. If the bulletin board entry's reencrypted vote is the same as the reencrypted vote sent by both voter process executions, then unless the underlying vote for each of these is also the same, individual verifiability is not satisfied. Likewise, if both designated verifier reencryption proofs are valid for the same reencrypted vote from the bulletin board, then unless the underlying vote is the same for both, universal verifiability is not satisfied: see Figure 7.8.

Figure 7.6 The *Voter* process in our verifiability model

```
let voter(keychannel:channel, votechannel:channel, pk_t1:pubkey, pk_t2:pubkey,
          pk_j:pubkey) =
 new i:bitstring;
 in(keychannel, sk_voter:secretkey);
 let pk_voter:pubkey = pk(sk_voter) in
 out(pch, pk_voter);
 in(pch, (delta:bitstring, dvsig:bitstring));
 if dvverify(dvsig, pk_t1, sk_voter) ≠ delta then
  0
     else
     new alpha:exponent;
          in(votechannel, vote:bitstring);
     let encvote = encrypt(vote, alpha, pk_t2, three) in
     new phi:exponent;
     let t1msg = encrypt((encvote, delta, pk_voter), phi, pk_t1, three) in
          out(ch, t1msg);
          in(ch, enct1reply:bitstring);
     let (reenc:bitstring, dvrp_a:bitstring) = decrypt(enct1reply, sk_voter) in
     if dvrpverify(dvrp_a, encvote, reenc, sk_voter, pk_t1) = true then
                    in(bulletinboard, signedbbentry:bitstring);
                    let (reenc_received:bitstring, vote_and_encrypted_ID:bitstring,
                              signedhash:bitstring) =
                                        verify(signedbbentry, pk_t2) in
                    ((if reenc_received = reenc then
                                  event pass(true);  (* Result is individually verifiable *)
                                  out(ch_pass, (true, reenc_received, i));
                                  out(verifchannel, (sk_voter, dvrp_a, encvote, reenc, vote))
                     )
                    | (if reenc_received = reenc then
                       if verify(signedhash, pk_t1) ≠ hash(vote_and_encrypted_ID) then
                         event fail(true) (* Result is *not* universally verifiable *)
                    )).
```

Figure 7.7 The main process in our verifiability model

```
process
 new sk_t1:secretkey; let pk_t1:pubkey = pk(sk_t1) in out(ch, pk_t1);
 new sk_t2:secretkey; let pk_t2:pubkey = pk(sk_t2) in out(ch, pk_t2);
 new sk_j:secretkey; let pk_j:pubkey = pk(sk_j) in out(ch, pk_j);

 ( new votechannel:channel;
  (
   (!new keychannel:channel; (voter(keychannel, votechannel, pk_t1, pk_t2, pk_j)
         | keygenerator(keychannel))
   ) | votegenerator(votechannel) | verifiabilitychecker(votechannel, pk_t2, pk_t1)
  ) |!t1(pk_t1, sk_t1, pk_t2, pk_j) |!t2(pk_t2, sk_t2, pk_t1, pk_j) | duplicatechecker()
 )
```

The process *duplicatechecker* works by reading in any two messages from two voter process executions, on a special private channel ch_pass. If the reencryptions sent with both of these messages are the same, then if the i value sent as the third part of the message is not the same, a duplicate vote has been recorded, and individual verifiability cannot hold.

ProVerif successfully terminates with the model, claiming that there are no executions of the event fail (i.e., it is unreachable), and that the event pass is reachable. This leads us to infer that our protocol is both universally and individually verifiable.

7.5.2 Correspondences

We adopt the work of Backes et al. (2008) in order to prove soundness (i.e., uniqueness, eligibility and inalterability) of our protocol. The authors formalise an *election process* as an unbounded number of voters and trusted authorities, running in parallel and sharing some secrets. Voters can be *honest* (always behaving according to specification), *corrupted* (registering correctly but then outputting all secrets) or *ad-hoc* (behaving arbitrarily between these two) (Backes et al., 2008, p. 196). The authors define an election process EP as a closed plain process:

$$EP \equiv \nu\tilde{n}.(!V^{hon}|!V^{cor}|V_{id_1}|\ldots|V_{id_k}|ID|A_1|\ldots|A_m)$$

where $\nu\tilde{n}$ represents a sequence of name restrictions (which are secret), such that

Figure 7.8 Disproving universal verifiability

```
let verifiabilitychecker(votechannel:channel, pk_t2:pubkey, pk_t1:pubkey) = (* P *)
  in(verifchannel, (skv1:secretkey, dvrp1:bitstring, encvote1:bitstring, reenc1:bitstring,
                    vote1:bitstring));
  in(verifchannel, (skv2:secretkey, dvrp2:bitstring, encvote2:bitstring, reenc2:bitstring,
                    vote2:bitstring));
  in(bulletinboard, signedbbentry:bitstring);
  let (reencvote:bitstring, vote_and_encrypted_ID:bitstring, signedhash:bitstring) =
      verify(signedbbentry, pk_t2) in
        (
          (if reencvote = reenc1 then
                  if reencvote = reenc2 then
                          if vote1 ≠ vote2 then event fail(true)
        ) |
         (if dvrpverify(dvrp1, encvote2, reencvote, skv1, pk_t1) = true then
                  if dvrpverify(dvrp2, encvote1, reencvote, skv2, pk_t1) = true then
                       if vote1 ≠ vote2 then event fail(true)
        )).
```

1. for a private channel $c_{id} \in \tilde{n}$ and two sequential contexts[1] V^{reg}, V^{vote}, $V^{hon} \equiv c_{id}(x_{id}).V^{reg}[\text{let } x_v \in \tilde{v} \text{ in } V^{vote}]$[2], where $\tilde{v}$ is the set of valid votes;

2. for the corrupted voters, there exists a process V^{κ} such that $V^{cor} \equiv c_{id}(x_{id}).V^{\kappa}$;

3. there exists a process ID', and a public channel $c_{id-pub} \notin \tilde{n}$ such that

$$ID \equiv (!\nu id.\overline{c_{id}}\langle id\rangle.\overline{c_{id-pub}}\langle id\rangle.\ let\ x_{id} = id\ in\ ID')$$
$$|\ let\ x_{id} = id_1\ in\ ID' | \ldots |\ let\ x_{id} = id_k\ in\ ID';$$

   All *id* values are not public and distinct, and the channel c_{id} is private. ID' is the remainder of the *ID* process, and contains no events.

4. there exists a public channel c_{votes}, a value $i \in [1, m]$, a variable x, a process P, and a context C such that

$$A_i \equiv C[\overline{c_{votes}}\langle x\rangle.P]$$

[1] A *sequential context* is defined as a plain context which includes no replication or parallel composition.

[2] Note that we have already defined the statement *let* $x = M$ *in* P as being equivalent to the substitution $\nu x.(\{^M/_x\}|P)$. The statement *let* $x_v \in \tilde{v}$ *in* V^{vote} should be treated analogously.

and c_{votes} occurs nowhere else in *EP*. Note that the index i of each administrator A_i is used only to define the number of administrators that are used.

(Backes et al., 2008, p. 197)

The honest voter, V^{hon}, registers and receives an identity on c_{id}, selects a valid vote and then casts it. Corrupted voters simply register and then become controlled by the adversary. The authors use an unbounded number of corrupted and honest voters. Ad-hoc voters v_{id_i} may or may not follow the protocol, and are not replicated. The authors also define an election context, *S*, that is a process with a hole which runs in parallel with the voters (Backes et al., 2008, p. 197).

Backes et al. define soundness in terms of correspondence assertions, where a causality relation among events within a protocol is imposed and tested. Events and execution traces are introduced to the applied pi calculus in Abadi et al. (2007). The authors annotate their election process *EP* with several events:

1. newid(*id*): occurs once the issuer has given an identity *id* to a voter
2. startid(*id*) and startcorid(*id*): occur when the registration phase begins for any voter (honest or otherwise)
3. beginvote(*id*,*v*): marks the start of the voting phase for a voter with ID *id*, casting vote *v*
4. endvote(*v*): occurs when the tallying of vote *v* has happened.

(Backes et al., 2008, p. 197)

Backes et al. therefore redefine an annotated version of *EP*:

$$EP \equiv \nu\tilde{n}.(!V^{hon}|!V^{cor}|V_{id_1}|\ldots|V_{id_k}|ID|A_1|\ldots|A_m)$$

such that

1. $V^{hon} \equiv c_{id}(x_{id}).\text{startid}(x_{id}).V^{reg}[let\ x_v \in \tilde{v}\ in\ \text{beginvote}(x_{id}, x_v).V^{vote}]$
2. $V^{cor} \equiv c_{id}(x_{id}).\text{startcorid}(x_{id}).V^{c}$

3. for each i, either $V_{id_i} \equiv \mathsf{startid}(id_i).V'_i$ where V'_i contains at most one $\mathsf{beginvote}(id_i, v)$ event, or $V_{id_i} \equiv \mathsf{startcorid}(id_i).V'_i$ where V'_i has no event

4. ID is defined as

$$ID \equiv (\nu id.\mathsf{newid}(id).\overline{c_{id}}\langle id\rangle.\overline{c_{id-pub}}\langle id\rangle.let\ x_{id} = id\ in\ ID') \mid$$
$$\mathsf{newid}(id_1).let\ x_{id} = id_1\ in\ ID' \mid \ldots \mid \mathsf{newid}(id_k).let\ x_{id} = id_k\ in\ ID'$$

   where $\mathsf{newid}(\cdot)$ does not occur elsewhere in EP, and ID' has no event

5. $A_i \equiv C[\mathsf{endvote}(x).\overline{c_{votes}}\langle x\rangle.P]$, and C, P, A_j for $j \neq i$ do not contain any event

(Backes et al., 2008, p. 198)

In order to define soundness, Backes et al. state that the following conditions must be satisfied. To illustrate the way in which the conditions are defined, a trace $t = t_1 :: \mathsf{someevent}(x) :: t_2 :: \mathsf{anotherevent}(x)$ is a trace in which the event someevent for a parameter x is triggered, *and* the event anotherevent is triggered, for the same x.

Definition 7.5.1. Soundness (Backes et al., 2008, p. 198)

As is standard in related literature, soundness is split into three categories: *inalterability* (i.e., that no-one can change anyone else's vote), *eligibility* (only eligible voters may vote) and *non-reusability/uniqueness* (every voter may only vote once). The definition of soundness which Backes et al. (2008) define encapsulates these three properties. A trace t guarantees soundness if and only if:

1. For any t_1, t_2, v such that $t = t1 :: \mathsf{endvote}(v) :: t_2$, there exist id, t', t'', t''' such that

   (a) $t_1 = t' :: \mathsf{startid}(id) :: t'' :: \mathsf{beginvote}(id, v) :: t'''$ and $t' :: t'' :: t''' :: t_2$ guarantees soundness

   (b) or $t_1 = t' :: \mathsf{startcorid}(id) :: t''$, and $t' :: t'' :: t_2$ guarantees soundness

2. For any t_1, t_2, id where $t = t_1 :: \mathsf{startid}(id) :: t_2$ or $t = t_1 :: \mathsf{startcorid}(id) :: t_2$, neither $\mathsf{startid}(id)$ nor $\mathsf{startcorid}(id)$ occur in $t_1 :: t_2$

3. For any t_1, t_2, id where $t = t_1 :: \mathsf{startid}(id) :: t_2$ or $t = t_1 :: \mathsf{startcorid}(id) :: t_2$, $\mathsf{newid}(id)$ occurs in t_1.

Note that all traces of the protocol must satisfy the above in order for the protocol to be sound.

Each of the three properties of soundness in Definition 7.5.1 are satisfied by the criteria above: *inalterability* is modelled by the notion that every vote v for every occurrence of $\mathsf{endvote}(v)$ is preceded by an occurrence of $\mathsf{beginvote}(id, v)$; i.e., every cast vote was cast by some voter. *Uniqueness* is modelled with an *injective* matching between $\mathsf{endvote}(v)$ and $\mathsf{beginvote}(id, v)$; i.e., there is exactly one occurrence of a vote being cast by a particular voter for every vote v. Finally, *eligibility* is tested by ensuring that there are no executions of $\mathsf{beginvote}(id, v)$ that were not preceded by newid (and as such, followed by $\mathsf{startid}(id)$) (Backes et al., 2008, p. 198).

7.5.2.1 Soundness in our Protocol

In order to verify that our protocol is sound, we need to ensure that Condition 1 of Definition 7.5.1 is satisfied. ProVerif can check this condition automatically, and our model for this check is to be found in the file `soundness.pv`, discussed further in Section 7.5.4. In order to test reachability properties such as those needed to assure soundness, one specifies the execution of an event with the line (for example) "**event** pkreceived(*pk_v*);". This event can then be tested for with the **query** command.

Of particular interest is the query

query k:pubkey, k':pubkey, v:bitstring;

$$\textbf{event}(\mathsf{voteCounted}(v)) \Longrightarrow$$
$$($$
$$(\textbf{inj-event}(\mathsf{beginVote}(k, v)) \Longrightarrow \textbf{inj-event}(\mathsf{pkreceived}(k)))$$
$$\vee$$
$$\textbf{inj-event}(\mathsf{corpkreceived}(k'))$$
$$).$$

which relates directly to Condition 1 above: it states that for any public key k or corrupted voter public key k', and for any vote v, if the event voteCounted(v) is triggered (by the tallier process), then either there was exactly one preceding execution of beginVote(k,v), which itself followed one execution of pkreceived(k), or there was exactly one execution of corpkreceived(k').

At a higher level, this means that for a vote to have been recorded by the talliers, either:

- that same vote must only have been cast once, by a voter who previously received a legitimate public key, or
- a corrupted voter received a public key, and then sent all information to the coercer (who presumably cast the vote).

On analysis of our model, ProVerif correctly terminates, stating that the query above is satisfied, proving that our protocol is sound (subject to the correctness of the model). The model of our protocol to check for soundness can be found in the file `soundness.pv`, discussed in Section 7.5.4.

7.5.3 Observational Equivalences

We again adopt the work of Backes et al. (2008) in order to prove a number of properties using observational equivalences. In their work, Backes et al. note that coercion resistance captures:

1. *Receipt-freeness*—a coercer cannot force a voter to prove how they have voted (via a receipt)
2. *Immunity to Simulation*—A voter cannot be forced to provide secrets required to impersonate her (since the coercer cannot determine if any secrets provided are real or fake)
3. *Immunity to Forced-abstention*—a coercer should not be able to tell whether a voter has voted or not (note that in the authors' work, this does not apply to an *in-person* coercer)

(Backes et al., 2008, p. 198)

The authors later demonstrate that coercion resistance implies both immunity to forced abstention and receipt-freeness. As a result, we focus on proving the four aspects of coercion resistance

defined by the authors in our work. As with Backes et al., we denote coercion resistance as the property that an attacker cannot distinguish between a voter that is coerced and provided legitimate secret material to him, and a cheating voter that gives him fake secrets and then participates in the vote normally. This means that the attacker cannot determine if or how a voter votes (as Backes et al. note, immunity to forced-abstention implies vote privacy, under the assumption that at least one other voter abstains from voting).

Based on their previous definition of an election process EP:

$$EP \equiv \nu\tilde{n}.(!V^{hon}|!V^{cor}|V_{id_1}|\ldots|V_{id_k}|ID|A_1|\ldots|A_m),$$

the authors define another voter process to represent a coerced voter, that registers as normal, then forwards all secrets to the coercer, and abstains:

$$V_i^{coerced(c)} \equiv let\ x_{id} = i\ in\ V^{reg}[\bar{c}\langle\tilde{u}\rangle],$$

and a plain context V^{fake} representing a strategy for a voter to cheat a coercer, by providing him with fake secrets:

$$V_i^{cheat(c)} \equiv let\ x_{id} = i\ in\ V^{reg}[let\ x_v = v'\ in\ V^{vote}|V^{fake}[\bar{c}\langle\tilde{u}\rangle]].$$

(Backes et al., 2008)

Backes et al. note that intuitively, an election context S is coercion-resistant if the context running with a coerced voter is observationally equivalent to the one with a cheating voter. However, on one side a vote is cast, and on the other one is not. Introducing a second voter V_j does not solve the problem:

$$S[V_i^{coerced(c)}|V_j(v')] \approx S[V_i^{cheat(c)}(v')|V_j^{abs}],$$

where V_j^{abs} represents a voter that registers then abstains, balances the number of votes but is still problematic. On the left hand side, the coercer receives real secrets; on the right, secrets that cannot be used to cast a vote. Given that this cannot be solved by assuming which way the

coercer will want to vote and balancing the votes in this way with a third voter, the authors adopt a different solution.

Instead of using a third voter, Backes et al. use an *extractor* process which takes the voter the coercer casts on behalf of V_i and tallies it directly. The extractor identifies the vote cast by the coercer, and on the right hand side casts it directly to the bulletin board, but on the left hand side abstains (equivalent to the voter voting with invalid secrets). This means that the coercer's vote is always balanced. The extractor is defined as a context

$$E_k^{c_1,c_2,z} = let\ x_{id} = k\ in\ V^{reg}[\nu\tilde{m}.(c_1(x).P_1|!c_2(y).P_2|C[if\ z \in \tilde{v}\ then[\ ]])]$$

for plain processes P_1, P_2, a sequential context C, and private channels between the coerced voter and tallying authority, c_1 and c_2 respectively. If the coercer votes, the variable z holds that vote. The result of using the extractor in the election processes is that coercion-resistance can be expressed as the observational equivalence between the following:

$$S'[V_i^{coerced(c,c_1)}|V_j(v')|E_k^{c_1,c_2,z}[0]] \approx S'[V_i^{cheat(c,c_1)}(v')|V_j^{abs}|E_k^{c1,c2,z}[\overline{c_{votes}}\langle z\rangle]].$$

In the first process, voter V_i complies with the coercer's demands, providing her secrets then abstaining. V_j votes v', and the extractor process simulates a voter nullifying her vote. In the second process, V_i cheats the coercer by providing incorrect secrets, and casts vote v' herself. V_j abstains, and the extractor tallies the vote that the coercer casts on behalf of V_i. Note that the process $V_i^{coerced(c,c_1)}$ is similar to the above $V_i^{coerced(c)}$, except that the coerced voter now outputs her secrets on channel c (for the coercer), and c_1 (for the extractor) (Backes et al., 2008, p. 200).

In order to capture coercion-resistance completely, Backes et al. define five criteria that an election context must satisfy. An election process S is coercion-resistant if:

1. there exists an election context S'' and two authorities A, A' such that $S \equiv S''[A|[\ ]]$, $S' \equiv \nu c_1, c_2.S''[A'|[\ ]]$, and $\nu c_2.(A'|!c_2(x)) \approx A$

2. $S'[V_i^{\not{coerced}(c,c_1)} | V_j(v') | E_k^{c_1,c_2,z}[0]] \approx S'[V_i^{\not{cheat}(c,c_1)}(v') | V_j^{abs} | E_k^{c_1,c_2,z}[\overline{c_{votes}}\langle z\rangle]]$ where $v' \in \tilde{v}$ is a valid vote

3. $\nu c.S'[!c(x) | V_i^{\not{cheat}(c,c_1)}(v') | V_j^{abs} | E_k^{c_1,c_2,z}[\overline{c_{votes}}\langle z\rangle]] \approx S[V_i(v') | V_j^{abs} | V_k^{abs}]$

4. Let $P = c(\tilde{x}).let\ x_v = v\ in\ V^{vote}\{^{\tilde{x}}/_{\tilde{u}}\}, v \in \tilde{v}, \tilde{u} = \mathsf{captured}(V^{reg})$, then

$$\nu c.S'[P | V_i^{\not{cheat}(c,c_1)}(v') | V_j^{abs} | E_k^{c_1,c_2,z}[\overline{c_{votes}}\langle z\rangle]] \approx$$
$$\nu c.S'[P | V_i^{\not{cheat}(c,c_1)}(v') | V_j^{abs} | E_k^{c_1,c_2,z}[\overline{c_{votes}}\langle v\rangle]]$$

5. $S[V_i^{inv-reg}] \approx \nu c_{votes}.(!c_{votes}(x) | S[V_i(v)])$ where v is a valid vote.

(Backes et al., 2008, p. 200)

As the authors further note, S' differs only from the original election context S in that the tallying authority outputs to the extractor in condition 1. Condition 2 represents the main equivalence for coercion resistance. Condition 3 notes that if the coercer, having been cheated, abstains from voting, then the extractor must abstain too; Condition 4 is that if the coercer casts a vote but using invalid credentials, the extractor must tally that vote directly. Note that for any context C, $\mathsf{captured}(C)$ defines the set of names and variables that are in scope for the hole in C. Condition 5 finally notes that votes cast with invalid secrets are ignored (Backes et al., 2008, p. 201). Backes et al. go on to define immunity to forced abstention:

$$S[V_i(v) | V_j^{abs}] \approx S[V_i^{abs} | V_j(v)],$$

i.e., an attacker cannot distinguish between two processes, in which voter a votes in the first process, and voter b in the second; and vote privacy:

$$S[V_i(v) | V_j(v')] \approx S[V_i(v') | V_j(v)].$$

The authors go on to prove that if S guarantees coercion resistance, then S is also immune to forced abstention attacks, and thence also implies vote privacy, under the assumption of a single

other abstaining voter (Backes et al., 2008, p. 201). They also state that under the assumption of an extractor being an acceptable abstraction of a third voter, coercion-resistance implies receipt-freeness.

7.5.3.1 Coercion Resistance, Privacy and Receipt-Freeness in Our Protocol

Backes et al. demonstrate their work by producing applied pi and ProVerif models for the last four of the above criteria for the JCJ protocol (Juels et al., 2005). As we have already stated, this protocol is of a similar form to ours, and hence we were able to write similar models in ProVerif to test for each part of coercion resistance.

Given the relative complexity of our protocol, we encountered a number of difficulties in modelling some parts of it. For example, note our original equational theory for re-encryption:

fun reencrypt(bitstring, exponent):bitstring.
equation forall *item*:bitstring, *alpha*:exponent, *beta*:exponent, *sk*:secretkey;
reencrypt(encrypt(*item*, *alpha*, pk(*sk*)), *beta*) = encrypt(*item*, sum(*alpha*, *beta*), pk(*sk*)).

It should be clear that the variable *item* can be replaced with an encryption itself, hence making (re-)encryptions of unbounded depth possible. In a complex protocol, this can cause non-termination in ProVerif. Our solution was to limit the number of permitted re-encryptions by including a series of re-encryption equations:

fun reencrypt(bitstring, exponent):bitstring.
equation forall *item*:bitstring, *alpha*:exponent, *beta*:exponent, *sk*:secretkey;
reencrypt(encrypt(*item*, *alpha*, pk(*sk*), three), *beta*) = encrypt(*item*, sum(*alpha*, *beta*), pk(*sk*), two).
reencrypt(encrypt(*item*, *alpha*, pk(*sk*), two), *beta*) = encrypt(*item*, sum(*alpha*, *beta*), pk(*sk*), one).
reencrypt(encrypt(*item*, *alpha*, pk(*sk*), one), *beta*) = encrypt(*item*, sum(*alpha*, *beta*), pk(*sk*), zero).

Although this limits the power of ProVerif, we feel it was a necessary and acceptable limitation: further re-encryptions could in no way give further power to the attacker (the only apparent aim

of further re-encryptions would be to somehow re-obtain the original ciphertext, which, given the encryption scheme we use, is mathematically and computationally highly improbable).

As noted earlier, observational equivalences are tested using the **choice** operator in ProVerif. Given that the models which test for coercion resistance are very long, we do not provide them here in depth, but instead give an example of our use of the **choice** operator in Figure 7.9.

Figure 7.9 The **choice** operator in use

```
let voterchoice(ch_voter_issuer:channel, ch_voter_registrar:channel, pk_t1:pubkey,
        pk_t2:pubkey, sk_vc:secretkey) =
    let pk_vc:pubkey = pk(sk_vc) in
    new noncevc:nonce;
    out(ch_voter_registrar, (n1, noncevc, pk_vc));
    in(ch_voter_registrar, (= n2, = noncevc, delta:bitstring, dvsig:bitstring));
    if dvverify(dvsig, pk_t1, sk_vc) = delta then
        in(chvote, vote:bitstring);
        new fakedelta:bitstring;
        let fakedvsig = fakedvsign(fakedelta, pk_t1, sk_vc) in
        new alpha:exponent;
        let encvote = encrypt(vote, alpha, pk_t2, three) in
        new phi:exponent;
        let t1msg = encrypt((encvote, choice[delta, fakedelta], pk_vc), phi,
                pk_t1, three) in
        out(comm, t1msg).
```

The example above models the choice of the voter to either vote using a genuine or fake δ value. Each of the models for coercion resistance terminate successfully. In each case, ProVerif claims that the suggested observational equivalence holds, leading us to infer that our protocol is coercion resistant, and, by extension, permits voter privacy, is not susceptible to forced abstention attacks, and is receipt-free, allowing for the abstraction of the third voter by the extractor.

7.5.4 Location of ProVerif Source Code

The ProVerif source files relating to the models and proofs presented in this chapter can be found at

`http://mattsmart.co/research/proverif`

At this address, six files can be found, which each relate to a different test:

Table 7.1 ProVerif Source Code Files

Filename	Description
`ivuv.pv`	Our model for individual and universal verifiability (Section 7.5.1.5)
`soundness.pv`	Our soundness model (Section 7.5.2.1)

`cr2.pv`
`cr3.pv`
`cr4.pv`
`cr5.pv`
	Four models for parts 2–5 of the coercion-resistance, receipt-freeness and privacy proofs (Section 7.5.3.1)

7.6 Summary

In this penultimate chapter, we have presented an extensive discussion of the applied pi calculus, and of the automated reasoning tool ProVerif, whose input may be in applied pi syntax. Using the abilities of ProVerif (namely, proving reachability and correspondence assertions, and selected observational equivalences), we have been able to prove that the protocol we discussed in Chapter 4 is universally and individually verifiable, sound, and provides voter privacy, receipt freeness and coercion-resistance.

Whenever using formal verification to prove the security properties of a protocol, one must ensure that certain problems are solved. Foremost, the (ProVerif) model must accurately represent the underlying protocol. In a language such as ProVerif, which only proves observational equivalences when using specific model formulations, and has no means for representing persistent storage, this is often a demanding requirement. Consequently, there is often a need to abstract away the complexities of a protocol (such as exactly *how* a designated verifier re-encryption proof works) from the model, allowing the basic purpose of the function to remain. A similar problem arises when formalising the requirements to be proved: how can one ensure that the formalisation of a requirement accurately captures what is meant?

At present, our work has focused on the modelling and verification of our first protocol, as discussed in Chapter 4. We are keen to extend our work on formalisation to our trusted computing-based protocol, as discussed in Chapter 5, and plan to consider this in future work.

8 Conclusions

In this chapter, we conclude the thesis. We begin by reviewing the results and contributions achieved by this thesis, and the lessons learned in pursuit of those results, and finish with a discussion of avenues for future work.

This thesis presents three key contributions:

- A concise, readily implementable protocol to provide remote electronic voting for United Kingdom national elections, allowing anonymity revocation in the case of personation;
- A remote electronic voting protocol which uses trusted computing (and Direct Anonymous Attestation) to assure authorities (and the voter) of the state of a remote voter's machine;
- A method by which remote users whose anonymity is revoked may be informed of this fact, generalised for many types of security discipline.

It seems fitting to close this thesis by repeating the statement with which it opened: designing electronic voting protocols is difficult. In the face of ever-increasing security requirements—conflicting with voters who wish to use an easily comprehensible system—producing a protocol which is attractive to both voters and security researchers alike is a formidable challenge.

In this thesis, we have striven to present work which satisfies this challenge. We began with the introduction of a protocol which, with minimal trust in a set of tellers who are not able to observe a vote, allows a voter to vote remotely, whilst unable to prove to a coercer whether how (or even if) they are voting. Our assumption, as with our later work, is simply that the voter can vote *once* unobserved, but an unlimited number of times otherwise, where her legitimate vote is the one counted. We introduced a novel use of designated verifier signatures, and an extension of earlier work to provide a proof of ballot validity for 1-out-of-L elections, to allow the voter to remain coercion-resistant.

The trust we placed in the first round of talliers in our first protocol was, we reasoned, the minimal required amount of trust for our protocol, given no further assumptions. However, we noted that this level of trust might be undesirable for a general election scheme. This prompted our work into use of the TPM and the DAA protocol for remote anonymous authentication of voter machines. In this work, we provided another novel method for voters to distinguish coerced from non-coerced votes, without the requirement for the user to generate designated verifier signatures. Our work is the first to utilise the TPM of a remote voter's machine to allow her to vote from home, whilst also assuring authorities of the state of the machine. This presents considerable avenues for future development, and a credible way in which to increase voter turnout.

In both of our first two protocols, we provide the ability for authorities to determine the identity of a voter from their ballot. This functionality is legally required in the United Kingdom. However, both protocols failed to alert the voter as to when her anonymity was revoked. We rectify this with our work in Chapter 6: we detail a generic protocol in which an anonymous user interacts with a remote server—proved to be trustworthy—to encrypt her identity, such that it can later be retrieved with sufficient authority. The nature of the deanonymisation is that the user is automatically able to determine whether her identity has been traced or not. This protocol is readily applicable to a number of security protocols, including that presented in Chapter 5. For this reason, we do not apply it directly to our electronic voting protocols, but the specifics of this application should be clear.

In our final chapter, we drew on the work of several authors on the formalisation of security protocols and their requirements in the applied pi calculus, a process calculus specifically designed for reasoning about security protocols. We used the automated reasoning tool ProVerif to formalise our work from Chapter 4, drawing on recent work on formal verification in e-voting to prove a number of security properties. We are, to our knowledge, the first to prove as many properties as we do in this manner.

8.1 Further Work

We consider our work on remote electronic voting to be deployable in the United Kingdom with a small amount of extra effort, and here discuss extra steps which could be addressed in the future.

Our work on the use of trusted computing in e-voting (Chapter 5) relies on the availability of a TPM in each voter's machine. At the moment, despite the increasing prevalence of TPMs in domestic computing environments, we are not yet 'there'. As service providers further demand secure communications with their users in the future, we envisage that the TPM will be included in more and more machines. Clearly, any deployment would need to consider *trust* in the makers of the TPM: a sensitive issue such as general elections means that one must be able to unquestionably trust any entity which could have access to secret data (a voter's vote, or identity). Manufacturers residing in countries with a questionable interest in UK electoral results would need to be carefully scrutinised. This aspect of future work is outside of the scope of this thesis, however.

An issue to consider for future work on trusted computing in e-voting is how to allow more than one voter to vote from a particular TPM—something which we do not currently consider. We anticipate that one suitable approach would be for each voter from a given TPM to complete a separate DAA join/sign protocol execution using fresh pseudonyms. However, allowing this whilst also preventing a voter from voting multiple times is complex, and we hence leave it as an interesting open question.

Another avenue for future work concerns our preliminary work on auditability of revocable

anonymity, discussed in Chapter 6. We feel that virtual monotonic counters provide a promising method for satisfying the 'digital envelope' problem, and would like to further explore the applicability of our auditability protocol to other security fields, such as digital cash and fair exchange/contract signing. As the work discussed in Chapter 6 was not the main focus of this thesis, we would like to spend further time formalising the security requirements of its area, and proving the protocol correct.

This leads us to further work on proving the correctness of our protocols. The main focus of this thesis is not on formal verification. However, we would like to formally prove the security of our second protocol. Though work has been done on formal verification of the DAA protocol, combining this with proving the security of our work is challenging. One must also bear in mind the issue that, as the basic applied pi calculus and ProVerif do not cater for protocols which are concerned with persistent state, an alternative approach to formalisation must be considered.

Our next step is to trial a small-scale implementation of the e-voting protocols we have developed, using a TPM simulator for the protocol in Chapter 5. We envisage that the contributions in this thesis will aid work on the deployment of national electronic voting schemes in countries concerned with remote voting, and particularly in the UK.

List of References

Martín Abadi and Cédric Fournet. Mobile Values, New Names, and Secure Communication. In *28th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, pages 104–115, London, 2001. ACM Press.

Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):36–47, 1999.

Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just Fast Keying in the Pi Calculus. *ACM Transactions on Information and System Security*, 10(3):9, 2007.

King Ables and Mark D. Ryan. Escrowed Data and the Digital Envelope. In *Trust and Trustworthy Computing. Proceedings Third International Conference, TRUST 2010*, pages 246–56. Springer Verlag, 2010.

Ben Adida. Helios: Web-based Open-Audit Voting. In *Proceedings, 17th USENIX Security Symposium*, 2008.

Ben Adida and Ronald L. Rivest. Scratch & Vote: Self-Contained Paper-based Cryptographic Voting. In *WPES '06: Proceedings of the 5th ACM workshop on Privacy in electronic society*, pages 29–40, New York, 2006. ACM.

Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Electing a University President using Open-Audit Voting: Analysis of real-word use of Helios. In *Proceedings, 2009 Electronic Voting Technology/Workshop on Trustworthy Elections*, 2009.

Ross Anderson, Mike Bond, and Stephen Murdoch. Chip and Spin. Technical report, Computer Laboratory, University of Cambridge, 2005.

Andrew Appel. How to defeat Rivest's ThreeBallot Voting System, 2006. URL http://www.cs.princeton.edu/~appel/papers/DefeatingThreeBallot.pdf.

Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-calculus. In *Proceedings - 21st IEEE Computer Security Foundations Symposium, CSF 2008*, pages 195–209. IEEE, 2008.

Jonathan Bannet, David W. Price, Algis Rudys, Justin Singer, and Dan S. Wallach. Hack-a-vote: Security issues with electronic voting systems. *IEEE Security and Privacy*, 2(1):32–7, 2004.

Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical Multi-Candidate Election System. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 274–83, New York, 2001. ACM.

Josh Benaloh. Simple Verifiable Elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, pages 1–7, Berkeley, CA, 2006. USENIX.

Josh Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, pages 1–8, Berkeley, CA, 2007. USENIX Association.

Josh Benaloh and Dwight Tuinstra. Receipt-Free Secret-Ballot Elections (Extended Abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 544–53, Montreal, 1994. ACM.

Josh C Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *PODC '86: Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, pages 52–62, Calgary, 1986. ACM Press.

Robert Blackburn. *The Electoral System in Britain*. Macmillan, London, 1995.

Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 82–96, Los Alamitos, CA, 2001. IEEE.

Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Proceedings, 20th Annual IEEE Symposium on Logic in Computer Science*, pages 331–340, Los Alamitos, CA, 2005. IEEE.

Daniel Bochsler. Can Internet voting increase political participation?, 2010. URL http://www.eui.eu/Projects/EUDO-PublicOpinion/Documents/bochslere-voteeui2010.pdf.

Dan Boneh and Philippe Golle. Almost Entirely Correct Mixing with Applications to Voting. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 68–77, Washington DC, 2002. ACM.

Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct Anonymous Attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS '04*, pages 132–45. ACM, 2004.

Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. In G. Persiano S. Cimato, C. Galdi, editor, *Security in Communication Networks, Third International Conference*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–89, 2003.

Jan Camenisch, Ueli Maurer, and Markus Stadler. Digital Payment Systems with Passive Anonymity-Revoking Trustees. *Journal of Computer Security*, 5(1):69–89, 1997.

Card Technology Today. Barclaycard on-the-pulse with three-in-one card. *Card Technology Today*, 19(7–8):3, July 2007.

David Challener, Kent Yoder, Ryan Catherman, David Safford, and Leendert Van Doorn. *A Practical Guide to Trusted Computing*. IBM Press, Boston, MA, 2008.

Chin-Chen Chang and Jung-San Lee. An anonymous voting mechanism based on the key exchange protocol. *Computers & Security*, 25(4):307–14, 2006.

David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

David Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology, Proceedings of Crypto '82*, pages 199–203, New York, 1982. Plenum.

David Chaum. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

David Chaum. Elections with Unconditionally-Secret Ballots and Disruption Equivalent to Breaking RSA. In *Advances in Cryptology—EUROCRYPT '88. Workshop on the Theory and Application of Cryptographic Techniques. Proceedings*, pages 177–82, Berlin, 1988. Springer-Verlag.

David Chaum. Secret-Ballot Receipts: True Voter-Verifiable Elections. *Security & Privacy*, 2(1): 38–47, 2004.

David Chaum and T. P. Pedersen. Wallet Databases with Observers. In *Advances in Cryptology - CRYPTO '92. 12th Annual International Cryptology Conference Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.

David Chaum, Amos Fiat, and Moni Naor. Untraceable Electronic Cash (Extended Abstract). In *Advances in Cryptology - CRYPTO '88*, pages 319–327, Berlin, 1990. Springer-Verlag.

David Chaum, Peter Y. A. Ryan, and Steve Schneider. A Practical, Voter-verifiable Election Scheme. In *Computer Security — ESORICS 2005 10th European Symposium on Research in Computer Security*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–39, Milan, September 2005. Springer-Verlag.

David Chaum, Jeroen van de Graaf, Peter Y. A. Ryan, and Poorvi L. Vora. High Integrity Elections. Cryptology ePrint Archive, Report 2007/270, 2007.

David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity II: end-to-end ver-

ifiability for optical scan election systems using invisible ink confirmation codes. In *Proceedings of the 2008 Conference on Electronic Voting Technology*, pages 1–13. USENIX, 2008a.

David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. Scantegrity: End-to-End Voter-Verifiable Optical-Scan Voting. *IEEE Security & Privacy*, 6(3):40–6, 2008b.

Yalin Chen, Jue-Sam Chou, Hung-Min Sun, and Ming-Hsun Cho. A Novel Electronic Cash System with Trustee-Based Anonymity Revocation From Pairing. *Electronic Commerce Research and Applications*, 2011. DOI: 10.1016/j.elerap.2011.06.002.

Yu-Yi Chen, Jinn-Ke Jan, and Chin-Ling Chen. The design of a secure anonymous Internet voting system. *Computers & Security*, 23(4):330–7, 2004.

B. Chevallier-Mames, P.A. Fouque, D. Pointcheval, J. Stern, and J. Traore. On Some Incompatible Properties of Voting Schemes. In *Proceedings of the IAVoSS Workshop on Trustworthy Elections*, 2006.

Sherman S. M. Chow, Joseph K. Liu, and Duncan S. Wong. Robust Receipt-Free Election System with Ballot Secrecy and Verifiability. In *Proceedings, 16th Annual Network & Distributed System Security Symposium*, San Diego, 2008.

Michael J. Ciaraldi, David Finkel, and Craig E. Wills. Risks in Anonymous Distributed Computing Systems. In *INC 2000. Proceedings of the 2nd International Network Conference*, pages 193–200, Plymouth, UK, 2000. University of Plymouth.

Joris Claessens, Claudia Diaz, Caroline Goemans, Bart Preneel, Joos Vandewalle, and Jos Dumortier. Revocable anonymous access to the internet? *Internet Research: Electronic Networking Applications and Policy*, 13(4):242–58, 2003.

Jeremy Clark and Urs Hengartner. Panic Passwords: Authentication Under Duress. In *Proceedings, 3rd USENIX Workshop on Hot Topics in Security, HotSec '08*. USENIX Association, 2008.

Jeremy Clark and Urs Hengartner. Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance. In *Proceedings, 15th International Conference on Financial Cryptography*, 2011.

Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. In *Proceedings, 2008 IEEE Symposium on Security and Privacy*, pages 354–68. IEEE, 2008.

R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Proceedings, 14th Annual International Cryptology Conference: CRYPTO '94*, pages 174–87. Springer, 1994.

Ronald Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-Authority Secret-Ballot Elections with Linear Work. In *Advances in Cryptology - EUROCRYPT'96. Proceedings*, pages 72–83, Berlin, 1996. Springer-Verlag.

Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *Advances in Cryptology - EUROCRYPT '97. Proceedings*, pages 103–18, Berlin, 1997. Springer-Verlag.

Lorrie Faith Cranor. Electronic Voting: Computerized polls may save money, protect privacy. *ACM Crossroads*, 2(4):12–16, 1996.

Lorrie Faith Cranor. Voting After Florida: No Easy Answers. *Ubuquity*, 2001(3), February 2001.

Lorrie Faith Cranor and Ron J. Cytron. Sensus: A Security-Conscious Electronic Polling System for the Internet. In *Proceedings of the Thirtieth Hawaii International Conference on System Sciences*, volume 3, pages 561–70, Wailea, Hawaii, 1997. IEEE.

Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6): 371–85, 2010.

George Davida, Yair Frankel, Yiannis Tsiounis, and Moti Yung. Anonymity Control in E-Cash

Systems. In *Proceedings of the 1997 1st International Conference on Financial Cryptography, FC'97*, volume 1318 of *Lecture Notes in Computer Science*, pages 1–16, Anguilla, 1997.

Stephanie Delaune, Steve Kremer, and Mark Ryan. Coercion-Resistance and Receipt-Freeness in Electronic Voting. In *Proceedings - 19th IEEE Computer Security Foundations Workshop, CSFW 2006*, pages 28–42, Venice, Italy, 2006. IEEE.

David L. Dill and Daniel Castro. The U.S. Should Ban Paperless Electronic Voting Machines. *Communications of the ACM*, 51(10):29–33, 2008.

David L. Dill, Bruce Schneier, and Barbara Simons. Voting and technology: who gets to count your vote? *Communications of the ACM*, 46(8):29–31, August 2003.

Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: the second-generation onion router. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 21–38. USENIX Association, 2004.

Gianluca Dini. A secure and available electronic voting service for a large-scale distributed system. *Future Generation Computer Systems*, 19(1):69–85, January 2003.

Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–72, 1985.

Aleks Essex, Jeremy Clark, Rick Carback, and Stefan Popoveniuc. Punchscan in Practice: an E2E Election Case Study. In *Proceedings, Workshop on Trustworthy Elections (WOTE)*, 2007.

Chun-I Fan and Yu-Kuang Liang. Anonymous Fair Transaction Protocols Based on Electronic Cash. *International Journal of Electronic Commerce*, 13(1):131–51, 2008.

Chun-I Fan and Wei-Zhe Sun. An efficient multi-receipt mechanism for uncoercible anonymous electronic voting. *Mathematical and Computer Modelling*, 48:1611–27, 2008.

A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proceedings, Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, New York, 1986. Springer-Verlag.

Russell A. Fink, Alan T. Sherman, and Richard Carback. TPM meets DRE: reducing the trust base for electronic voting using trusted platform modules. *IEEE Transactions on Information Forensics and Security*, 4(4):628–37, 2009.

Kevin Fisher, Richard Carback, and Alan Sherman. Punchscan: Introduction and System Definition of a High-Integrity Election System. In *Proceedings, 2006 Workshop on Trustworthy Elections*, 2006.

David Foster, Laura Stapleton, and Huirong Fu. Secure Remote Electronic Voting. In *Proceedings, 2006 IEEE International Conference on Electro Information Technology*, pages 591–6, NJ, 2006. IEEE.

P. A. Fouque, G. Poupard, and J. Stern. Sharing Decryption in the Context of Voting or Lotteries. In *Proceedings, Financial Cryptography 2000*, volume 1962 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2000.

Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Transferable Constant-Size Fair E-Cash. In Juan Garay, Atsuko Miyaji, and Akira Otsuka, editors, *Cryptology and Network Security*, volume 5888 of *Lecture Notes in Computer Science*, pages 226–247. Springer, 2009.

Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A Practical Secret voting Scheme for Large Scale Elections. In *Advances in Cryptology - AUSCRYPT '92. Workshop on the Theory and Application of Cryptographic Techniques Proceedings*, pages 244–51, Berlin, 1993. Springer-Verlag.

Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal re-encryption for mixnets. In *Topics in Cryptology - CT-RSA 2004. Cryptographers' Track at the RSA Conference 2004*, pages 163–78, 2004.

Yong Guan, Xinwen Fu, Riccardo Bettati, and Wei Zhao. An Optimal Strategy for Anonymous Communication Protocols. In *Proceedings, 22nd International Conference on Distributed Computing Systems*, pages 257–66, Los Alamitos, 2002. IEEE.

L. Harn. Group-Oriented (t, n) Threshold Digital Signature Scheme and Digital Multisignature. In *IEE Proceedings—Computers and Digital Techniques*, volume 141, pages 307–13, 1994.

Mark Herschberg. Secure Electronic Voting Over the World Wide Web. Master's thesis, Department of Electrical Engineering and Computer Science, MIT, 1997.

Alejandro Hevia and Marcos Kiwi. Electronic jury voting protocols. *Theoretical Computer Science*, 321(1):73–94, 2004.

Martin Hirt. *Multi-Party Computation: Efficient Protocols, General Adversaries and Voting*. PhD thesis, ETH Zurich, 2001.

Martin Hirt. Receipt-Free K-out-of-L Voting Based on ElGamal Encryption. In *Towards Trustworthy Elections*, volume 2000 of *Lecture Notes in Computer Science*, pages 64–82. Springer, 2010.

Martin Hirt and Kazue Sako. Efficient Receipt-Free Voting Based on Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2000. Proceedings*, volume 1807 of *Lecture Notes in Computer Science*, pages 539–56, Bruges, Belgium, 2000. Springer-Verlag.

C. A. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.

Philippe Holle, Sheng Zhong, Dan Boneh, Markus Jakobsson, and Ari Juels. Optimistic Mixing for Exit Polls. In *Advances in Cryptology—ASIACRYPT 2002. 8th International Conference on the Theory and Application of Cryptology and Information Security. Proceedings*, volume 2501 of *Lecture Notes in Computer Science*, pages 451–65, Berlin, 2002. Springer-Verlag.

X. Hou and C. H. Tan. On Fair Traceable Electronic Cash. In *Proceedings, 3rd Annual Communication Networks and Services Research Conference*, pages 39–44. IEEE, 2005.

Markus Jakobsson. A practical mix. In *Advances in Cryptology—EUROCRYPT '98. International Conference on the Theory and Application of Cryptographic Techniques. Proceedings*, pages 448–61, Berlin, 1998. Springer-Verlag.

Markus Jakobsson and Ari Juels. Mix and Match: Secure Function Evaluation via Ciphertexts (Extended Abstract). In *Advances in Cryptology—ASIACRYPT 2000. 6th International Conference on the Theory and Application of Cryptology and Information Security. Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–77, Berlin, 2000. Springer-Verlag.

Markus Jakobsson and Moti Yung. Revokable and Versatile Electronic Money (Extended Abstract). In *CCS '96: Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pages 76–87, New York, 1996. ACM Press.

Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated Verifier Proofs and Their Applications. In *Proceedings, EUROCRYPT '96*, pages 143–154. Springer-Verlag, 1996.

Markus Jakobsson, David M'Raihi, Yiannis Tsiounis, and Moti Yung. Electronic Payments: Where Do We Go From Here? In R. Baumgard, editor, *Secure Networking - CQRE [Secure] '99*, volume 1740 of *Lecture Notes in Computer Science*, pages 43–63. Springer-Verlag, 1999.

Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making Mix Nets Robust for Electronic Voting by Randomised Partial Checking. In *Proceedings of the 11th USENIX Security Symposium*, pages 339–53, Berkeley, 2002. USENIX Assoc.

David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. A Security Analysis of the Secure Electronic Registration and Voting Experiment (SERVE), 2004. URL `http://www.servesecurityreport.org`.

Hugo Jonker and Wolter Pieters. Anonymity in voting revisited. In *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 216–230. Springer-Verlag, 2010.

Andreu Riera Jorba, José Antonia Ortega Ruiz, and Paul Brown. Advanced Security to Enable Trustworthy Electronic Voting. In *Proceedings, Third European Conference on E-Government*, Dublin, Ireland, 2003. EJEG.

Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-Resistant Electronic Elections. In

WPES'05: Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, pages 61–70, New York, 2005. ACM.

Aggelos Kiayias and Moti Yung. Self-Tallying Elections and Perfect Ballot Secrecy. In *Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 141–158, 2002.

Aggelos Kiayias and Moti Yung. The Vector-Ballot e-Voting Approach. In *Proceedings, Financial Cryptography 2004*, pages 72–89. Springer, 2004.

Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan Wallach. Analysis of an Electronic Voting System. In *Proceedings, 2004 IEEE Symposium on Security and Privacy*, pages 27–40, 2004.

Steve Kremer and Mark D. Ryan. Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In *Proceedings of the European Symposium on Programming (ESOP'05)*, volume 3344 of *Lecture Notes in Computer Science*, pages 186–200, Berlin, 2005. Springer-Verlag.

Steve Kremer, Mark Ryan, Ben Smyth, and Mounira Kourjieh. Towards Automatic Analysis of Election Verifiability Properties. In *Proceedings, Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10)*, volume 6186 of *Lecture Notes in Computer Science*, pages 165–82. Springer-Verlag, 2010.

Dennis Kügler and Holger Vogt. Auditable Tracing with Unconditional Anonymity. In *Proceedings of the Second International Workshop on Information Security Applications: WISA 2001*, pages 151–163, 2001.

Dennis Kügler and Holger Vogt. Fair tracing without trustees. In *Financial Cryptography. 5th International Conference, FC 2001. Proceedings*, volume 2339 of *Lecture Notes in Computer Science*, pages 136–48, Berlin, 2002. Springer-Verlag.

Dennis Kügler and Holger Vogt. Off-line Payments with Auditable Tracing. In *Financial Cryptography. 6th International Conference, FC 2002. Revised Papers*, volume 2357 of *Lecture Notes in Computer Science*, pages 269–81, Berlin, 2003. Springer.

Byoungcheon Lee and Kwangjo Kim. Receipt-free Electronic Voting through Collaboration of Voter and Honest Verifier. In *Proceedings, JW-ISC 2000*, pages 101–108, 2000.

Byoungcheon Lee and Kwangjo Kim. Receipt-Free Electronic Voting Scheme with a Tamper-Resistant Randomizer. In *Proceedings of ICISC2002*, pages 389–406. Springer-Verlag, 2002.

Byoungcheon Lee, Colin Boyd, Kwangjo Kim, Jeongmo Yang, and Seungjae Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *In Proc. of Information Security and Cryptology (ICISC'03)*, volume 2971 of *Lecture Notes in Computer Science*, pages 245–58. Springer-Verlag, 2004.

Horng-Twu Liaw. A secure electronic voting protocol for general elections. *Computers & Security*, 23(2):107–19, 2004.

Local Government Association. The Implementation of Electronic Voting in the UK, 2002. URL http://www.communities.gov.uk/documents/localgovernment/pdf/133544.pdf.

Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym Systems. In *Selected Areas in Cryptography: 6th Annual International Workshop, SAC'99*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–99, Kingston, Ontario, 2000. Springer-Verlag.

Ülle Madise and Tarvi Martens. E-voting in Estonia 2005. The first practice of country-wide binding Internet voting in the world. In *Electronic Voting 2006, 2nd International Workshop: Proceedings*, volume P-86 of *Lecture Notes in Informatics*, pages 15–26, Bregenz, Austria, 2006. Gesellschaft für Informatik.

Gary T. Marx. What's in a Name? Some Reflections on the Sociology of Anonymity. *The Information Society*, 15:99–112, 1999.

Rebecca Mercuri. A Better Ballot Box? *IEEE Spectrum*, 39(10):46–50, October 2002.

Markus Michels and Patrick Horster. Some remarks on a receipt-free and universally verifiable Mix-type voting scheme. In *Advances in Cryptology - ASIACRYPT'96 International Converence*

on the Theory and Applications of Cryptology and Information Security, pages 125–32, Berlin, 1996. Springer-Verlag.

Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

Robin Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Parts I and II. *Information and Computation*, 100(2):1–46 *(I)*; 1–41 *(II)*, 1992.

Tal Moran and Moni Naor. Basing Cryptographic Protocols on Tamper-Evident Seals. *Theoretical Computer Science*, 411(10), 2010.

Moni Naor and Adi Shamir. Visual Cryptography. In *Proceedings, Advances in Cryptology (EUROCRYPT '94)*, volume 950 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, 1995.

National Institute of Standards and Technology. *Digital Signature Standard (DSS): FIPS PUB 186-3*. Gaithersburg, MD, 2009.

C. Andrew Neff. Election Confidence: A Comparison of Methodologies and Their Relative Effectiveness at Achieving It, 2003.

C. Andrew Neff. Practical High Certainty Intent Verification for Encrypted Votes. VoteHere, 2004.

Open Rights Group. Electronic Voting: A Challenge to Democracy?, 2007. URL `http://www.openrightsgroup.org/wp-content/uploads/org-evoting-briefing-pack-final.pdf`.

Pascal Paillier. Public-Key Cryptosystems Based on Discrete Logarithms Residues. In *Proceedings, Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*. Springer, 1999.

Nathanael Paul and Andrew S. Tanenbaum. Trustworthy Voting: From Machine to System. *Computer*, 42(5):35–41, 2009.

T. P. Pedersen. A Threshold Cryptosystem Without a Trusted Party. In *Proceedings, Advances in Cryptology, EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–6. Springer-Verlag, 1991.

T. P. Pedersen. Non-interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Proceedings, Advances in Cryptology (CRYPTO '91)*, volume 576 of *Lecture Notes in Computer Science*, pages 129–40. Springer Verlag, 1992.

David Pointcheval. Self-Scrambling Anonymizers. In Y. Frankel, editor, *Proceedings of Financial Cryptography 2000*, Berlin, 2000. Springer-Verlag.

Stefan Popoveniuc and Ben Hosp. An Introduction to Punchscan. In *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 242–59. Springer, 2010.

Brian Randell and Peter Y.A. Ryan. Voting Technologies and Trust. Technical Report CS-TR: 911, Newcastle University, May 2005.

Indrajit Ray, Indrakshi Ray, and Natarajan Narasimhamurthi. An Anonymous Electronic Voting Protocol for Voting Over The Internet. In *Proceedings, Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems. WECWIS 2001*, pages 188–90, San Juan, CA, 2001. IEEE.

Michael K. Reiter and Aviel D. Rubin. Anonymous Web Transactions with Crowds. *Communications of the ACM*, 42(2):32–38, 1999.

Ronald Rivest and Warren Smith. Three Voting Protocols: ThreeBallot, VAV, and Twin. In *Proceedings of Electronic Voting Technology Workshop, 2007*, pages 1–14, Boston, MA, 2007.

Peter Y. A. Ryan and Vanessa Teague. Pretty Good Democracy. In *Proceedings of the 17th Workshop on Security Protocols*, 2009.

Peter Y.A. Ryan. Prêt à Voter With a Human-Readable, Paper Audit Trail. Technical Report CS-TR: 1038, Newcastle University, 2007.

Peter Y.A. Ryan and S.A. Schneider. Prêt à Voter with re-encryption mixes. In *Computer Security—ESORICS 2006. Proceedings*, volume 4189 of *Lecture Notes in Computer Science*, pages 313–26, Hamburg, 2006. Springer-Verlag.

Shahrokh Saeednia, Steve Kremer, and Olivier Markowitch. An Efficient Strong Designated Verifier Signature Scheme. *Information Security and Cryptology — ICISC*, 2971:40–54, 2003.

Kazue Sako and Joe Kilian. Receipt-Free Mix-Type Voting Scheme: A practical solution to the implementation of a voting booth. In *Advances in Cryptology - EUROCRYPT'95. Proceedings*, pages 393–403, Berlin, 1995. Springer-Verlag.

Luis F.G. Sarmenta, Marten van Dijk, Charles W. O'Donnell, Jonathan Rhodes, and Srinivas Devadas. Virtual Monotonic Counters and Count-Limited Objects using a TPM without a trusted OS. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing, STC'06*, pages 27–42, New York, 2006. ACM.

Steve Schneider, Sriramkrishnan Srinivasan, Chris Culnane, James Heather, and Zhe Xia. Prêt à Voter with Write-Ins. Technical report, University of Surrey, 2011.

Bruce Schneier. *Applied Cryptography*. Wiley, New York, 1996.

Berry Schoenmakers. A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting. In *Advances in Cryptology - CRYPTO'99. 19th Annual International Cryptology Conference*, pages 148–64, Berlin, 1999. Springer-Verlag.

A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–3, 1979.

D. Shanks. Class Number, a Theory of Factorization and Genera. In *Proceedings, Symposium on Pure Mathematics, vol. 20*, pages 415–40. AMS, 1971.

Clay Shields and Brian Neil Levine. A Protocol for Anonymous Communication over the Internet. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 33–42. ACM, 2000.

Matt Smart and Eike Ritter. Remote Electronic Voting with Revocable Anonymity. In *Proceedings of the Fifth International Conference on Information Systems Security (ICISS '09)*, volume 5905 of *Lecture Notes in Computer Science*, pages 39–54, Kolkata, India, 2009. Springer.

Matt Smart and Eike Ritter. True Trustworthy Elections: Remote Electronic Voting Using Trusted Computing. In *Proceedings, 5th Benelux Workshop on Information and System Security (WISSec '10)*, 2010.

Matt Smart and Eike Ritter. True Trustworthy Elections: Remote Electronic Voting Using Trusted Computing. In *Proceedings, 8th International Conference in Autonomic and Trusted Computing (ATC-2011)*, volume 6906 of *Lecture Notes in Computer Science*, pages 187–200, Banff, Canada, 2011. Springer.

Matt Smart and Eike Ritter. Auditable Envelopes: Tracking Anonymity Revocation Using Trusted Computing. In *Proceedings, Fifth International Conference on Trust and Trustworthy Computing (TRUST 2012)*, volume 7344 of *Lecture Notes in Computer Science*, pages 19–33, Vienna, Austria, 2012. Springer.

W. Smith. New Cryptographic Election Protocol with Best-Known Theoretical Properties. In *Proceedings, Workshop on Frontiers in Electronic Elections (FEE 2005)*, Milan, 2005.

Olivier Spycher, Reto Koenig, Rolk Haenni, and Michael Schläpfer. A New Approach Towards Coercion-Resistant Remote E-Voting in Linear Time. In *Proceedings, 15th International Conference on Financial Cryptography*, 2011.

Emil Stefanov and Mikhail Atallah. Duress Detection for Authentication Attacks Against Multiple Administrators. In *Proceedings, 2010 ACM Workshop on Insider Threats*, pages 37–46. ACM, 2010.

Tim Storer and Ishbel Duncan. Polsterless Remote Electronic Voting. *Journal of e-Government*, 1 (1):75–103, 2004.

Tim Storer and Ishbel Duncan. Two Variations to the mCESG Pollsterless e-Voting Scheme.

In *Proceedings, 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*. IEEE, 2005.

Zuowen Tan. An Off-line Electronic Cash Scheme Based on Proxy Blind Signature. *The Computer Journal*, 54(4):505–512, 2011.

TPM Main: Part 1: Design Principles, Version 1.2, Revision 116. TCG: Trusted Computing Group, October 2011a. URL http://bit.ly/camUwE.

TPM Main: Part 2: Structures of the TPM, Version 1.2, Revision 116. TCG: Trusted Computing Group, October 2011b. URL http://bit.ly/camUwE.

TPM Main: Part 3: Commands, Version 1.2, Revision 116. TCG: Trusted Computing Group, October 2011c. URL http://bit.ly/camUwE.

Melanie Volkamer and Robert Krimmer. Secrecy forever? Analysis of Anonymity in Internet-based Voting Protocols. In *Proceedings, First International Conference on Availability, Reliability and Security, ARES 2006*, pages 340–347, Vienna, 2006. IEEE.

Melanie Volkamer, Ammar Alkassar, Ahmad-Reza Sadeghi, and Stefan Schulz. Enabling the Application of Open Systems like PCs for Online Voting. In *Proceedings of the 2006 Workshop on Frontiers in Electronic Elections—FEE'06*, 2006.

Sebastiaan von Solms and David Naccache. On Blind Signatures and Perfect Crimes. *Computer & Security*, 11(6):581–583, 1992.

Changji Wang and Rongbo Lu. An ID-based Transferable Off-Line e-Cash System with Revokable Anonymity. In *Proceedings, International Symposium on Electronic Commerce and Security, ISECS 2008*, pages 758–62. IEEE, 2008.

Stefan G. Weber and Max Mühlhäuser. *Multilaterally Secure Ubiquitous Auditing*, volume 329 of *Studies in Computational Intelligence*, chapter 10. Springer Verlag, first edition, 2011.

Stefan G. Weber, Roberto Araújo, and Johannes Buchmann. On Coercion-Resistant Electronic Elections with Linear Work. In *Proceedings, 2007 2nd International Conference on Availability, Reliability and Security*, pages 908–16, Vienna, 2007. IEEE.

Douglas Wikström. A Sender Verifiable Mix-Net and a New Proof of a Shuffle. In *Advances in Cryptology - ASIACRYPT 2005. 11th International Conference on the Theory and Application of Cryptology and Information Security*, volume 3788 of *Lecture Notes in Computer Science*, pages 273–92, 2005.

Z. Xia, S. Schneider, J. Heather, P. Ryan, D. Lundin, R. Peel, and P. Howard. Prêt à Voter: All-In-One. In *Proceedings of the 2007 Workshop on Trustworthy Elections—WOTE'07*, 2007.